LE
GUIDE
DU
CODEUR
XAML
Créez vos interfaces graphiques
pour Windows Vista™
Micro
Application

Copyright

© 2006 Micro Application
20-22, rue des Petits-Hôtels
75010 Paris

1ère Édition - Septembre 2006

Auteur

Jean-Alain BAEYENS

Avertissement aux utilisateurs

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (article L122-4 du code de la propriété intellectuelle).
Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles L335-2 et suivants du code de la propriété intellectuelle.
Le code de la propriété intellectuelle n'autorise aux termes de l'article L122-5 que les reproductions strictement destinées à l'usage privé et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et courtes citations dans un but d'exemple et d'illustration.

Les informations contenues dans cet ouvrage sont données à titre indicatif et n'ont aucun caractère exhaustif voire certain. A titre d'exemple non limitatif, cet ouvrage peut vous proposer une ou plusieurs adresses de sites Web qui ne seront plus d'actualité ou dont le contenu aura changé au moment où vous en prendrez connaissance.
Aussi, ces informations ne sauraient engager la responsabilité de l'Editeur. La société MICRO APPLICATION ne pourra être tenue responsable de toute omission, erreur ou lacune qui aurait pu se glisser dans ce produit ainsi que des conséquences, quelles qu'elles soient, qui résulteraient des informations et indications fournies ainsi que de leur utilisation.
Tous les produits cités dans cet ouvrage sont protégés, et les marques déposées par leurs titulaires de droits respectifs. Cet ouvrage n'est ni édité, ni produit par le(s) propriétaire(s) de(s) programme(s) sur le(s)quel(s) il porte et les marques ne sont utilisées qu'à seule fin de désignation des produits en tant que noms de ces derniers.

ISBN : 2-7429-6729-X

Couverture réalisée par Room22.

MICRO APPLICATION
20-22, rue des Petits-Hôtels
75010 PARIS
Tél. : 01 53 34 20 20
Fax : 01 53 34 20 00
http://www.microapp.com

Support technique
Également disponible sur
www.microapp.com

Retrouvez des informations sur cet ouvrage !

Rendez-vous sur le site Internet de Micro Application **www.microapp.com**. Dans le module de recherche, sur la page d'accueil du site, entrez la référence à 4 chiffres indiquée sur le présent livre. Vous accédez directement à sa fiche produit.

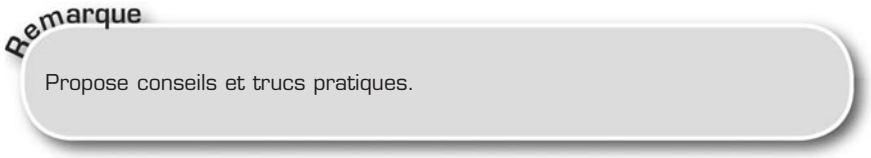
Avant-propos

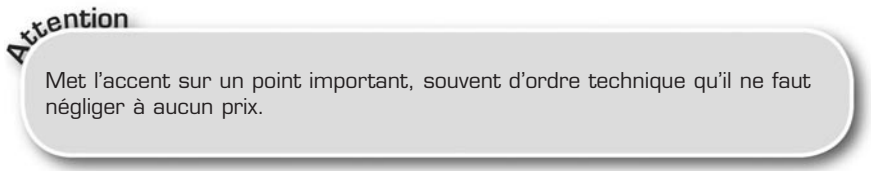
Le collection *Guide du codeur* s'adresse aux personnes initiées à la programmation qui souhaitent découvrir une technologie particulière. Sans négliger les aspects théoriques, nous donnons toujours priorité à la pratique afin que vous puissiez rapidement être autonome. Avant d'entrer dans le vif du sujet, notez ces quelques informations générales à propos de la collection.

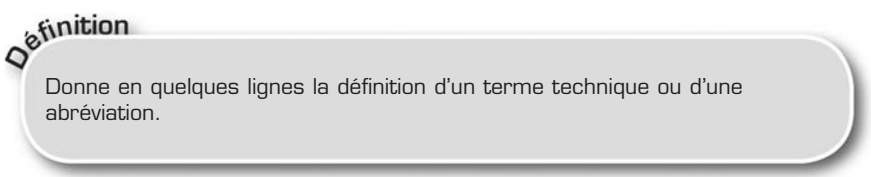
Conventions typographiques

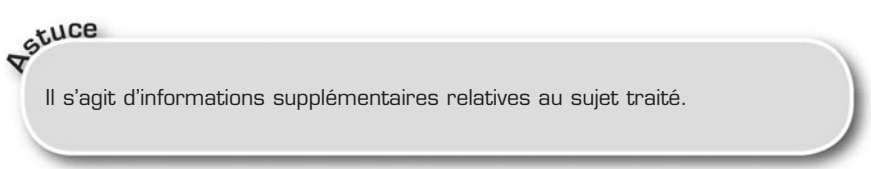
Afin de faciliter la compréhension de techniques décrites, nous avons adopté les conventions typographiques suivantes :

- **gras** : menu, commande, boîte de dialogue, bouton, onglet.
- *italique* : zone de texte, liste déroulante, case à cocher, bouton radio.
- `Police bâton` : instruction, listing, texte à saisir.
- ➥ : dans les programmes, indique un retour à la ligne dû aux contraintes de la mise en page.









Sommaire

1 Introduction 11

1.1. Avertissement 12
1.2. Prérequis 12
1.3. Présentation de XAML 13
Qu'est-ce que XAML ? 13
Petits rappels XML 14
Les principes généraux 15
1.4. Utiliser XAMLPad 17
1.5. Checklist 19

2 Fonctionnalités de base 21

2.1. Afficher du texte 22
Avec un Label 22
Avec un TextBlock 30
2.2. Introduire du texte 38
2.3. Créer un bouton 46
2.4. Afficher un cadre 47
2.5. Afficher une image 48
2.6. Checklist 53

3 Disposer les éléments à l'écran 55

3.1. Utiliser les coordonnées 56
3.2. Utiliser une grille 61
3.3. Mettre en page avec un WrapPanel 68
3.4. Utiliser un empilement 70
3.5. Utiliser le docking 72
3.6. Autoriser le défilement 77
3.7. Mélanger les techniques de mise en page 81
3.8. Créer une page composite 88
3.9. Checklist 90

4 Les autres contrôles de base 91

4.1. Créer une liste déroulante 92
4.2. Créer une ComboBox 98
4.3. Créer une case à cocher 100
4.4. Utiliser les boutons radio 102
4.5. Placer des info-bulles 106
4.6. Utiliser les panneaux à onglets 109

4.7. Créer un bouton automatique 112
4.8. Utiliser un Slider 114
4.9. Utiliser un Expander 118
4.10. Utiliser une ViewBox 121
4.11. Utiliser un Popup 123
4.12. Ajouter de la vidéo dans la fenêtre 126
4.13. Checklist 129

5 Créer une application 131
5.1. Créer une application Windows 132
5.2. Gérer les événements 138
5.3. Héberger une application dans un browser 140
Aperçu de cette technologie 140
La sécurité et les WBA 141
Héberger et exécuter ce type d'application 141
Quand recourir à ce modèle d'application ? 142
Créer une WBA 142
Enchaînement des pages 147
5.4. Les pages fonctions 149
5.5. Créer une application Windows navigable 157
5.6. Les applications avec WPF/E 165
5.7. Checklist 167

6 Les menus 169
6.1. Créer un menu 170
Le menu principal 170
Les sous-menus 171
Rendre un élément du menu inactif 172
Cocher un élément du menu 173
Associer une action à un menu 173
Rendre le menu dynamique 176
6.2. Créer un menu contextuel 178
6.3. Créer une barre d'outils 183
Une barre d'outils statique 183
Un ensemble de barres d'outils 185
6.4. Checklist 189

7 Lier les données à son interface utilisateur . . . 191
7.1. Lier les données à un DataSet 192

7.2. Lier les données à un objet métier 203
7.3. Lier les données sans utiliser le code .NET 207
7.4. Checklist 218

8 Fonctionnalités avancées 219
8.1. Appliquer des transformations sur les contrôles 220
8.2. Créer une ressource 223
8.3. Créer un style 227
Utiliser les triggers 238
Créer une animation 241
8.4. Checklist 247

9 Les documents 249
9.1. Utiliser FixedDocument 250
9.2. Utiliser FlowDocument 254
9.3. Éditer un document 275
9.4. Annoter un document 282
9.5. Checklist 288

10 Les outils graphiques 289
10.1. Le designer de Visual Studio (nom de code CIDER) 290
10.2. Dans la gamme expression 303
Graphic Designer 303
Interactive Designer 306
10.3. Aurora Designer 310
10.4. ZAM 3D 313
10.5. Checklist 314

11 Le dessin 315
11.1. Le dessin en 2D 316
11.2. Le dessin en 3D 323
11.3. Checklist 327

12 Réaliser une application complète 329
12.1. Checklist 349

13 Annexes 351
13.1. XAML sur le Web 352

13.2. Glossaire 359
13.3. Schéma d'héritage des différentes classes Visual 363
Schéma d'héritage des différentes classes Visual 363
Le détail de l'héritage dans la branche Control. 364
Schéma d'héritage des différentes classes ContentElement . . . 366
Schéma d'héritage des différentes classes Freezable 366
13.4. Résumé des classes et des attributs utilisés 368
Classe ArcSegment 368
Classe BezierSegment 368
Classe Border 368
Classe Button 369
Classe Canvas 370
Classe CheckBox 371
Classe ColorAnimation 373
Classe ComboBox 373
Classe DiffuseMaterial 374
Classe DirectionalLight 374
Classe DockPanel 375
Classe DocumentViewer 375
Classe DoubleAnimation 376
Classe DoubleAnimationUsingKeyFrames 376
Classe Ellipse 376
Classe EventTrigger 377
Classe Expander 377
Classe Figure 377
Classe FixedPage 378
Classe FixedDocument 379
Classe Floater 379
Classe FlowDocument 380
Classe GradientStop 380
Classe Grid 380
Classe GridSplitter 381
Classe GridView 382
Classe GridViewColumn 382
Classe Hyperlink 382
Classe Image 383
Classe ImageBrush 383
Classe Label 384
Classe Line 385
Classe LinearGradientBrush 385
Classe LineSegment 385
Classe ListBox 386
Classe ListView 387

Sommaire

Classe Menu 387
Classe MenuItem 387
Classe MeshGeometry3D 388
Classe NavigationWindow 388
Classe ObjectDataProvider 389
Classe Page 389
Classe PageContent 390
Classe Paragraph 391
Classe Path 392
Classe PathFigure 392
Classe Pen 392
Classe PerspectiveCamera 393
Classe Polygon 393
Classe Polyline 393
Classe PolylineSegment 393
Classe Popup 394
Classe RadialGradientBrush 394
Classe RadioButton 394
Classe Rectangle 395
Classe RotateTransform 396
Classe RepeatButton 396
Classe ScaleTransform 396
Classe ScrollViewer 396
Classe Section 397
Classe Setter 397
Classe SkewTransform 397
Classe Slider 398
Classe SolidColorBrush 399
Classe SplineDoubleKeyFrame 399
Classe StackPanel 399
Classe StoryBoard 400
Classe Style 400
Classe Table 401
Classe TableCell 401
Classe TableColumn 401
Classe TableRow 401
Classe TabControl 401
Classe TabItem 402
Classe TextBlock 403
Classe TextBox 404
Classe Toolbar 405
Classe ToolbarTray 405
Classe TranslateTransform 406

Classe TreeView 406
Classe TreeViewItem 406
Classe Trigger 406
Classe ViewBox 407
Classe Viewport3D 407
Classe Window 408
Classe WrapPanel 409
Classe XmlDataProvider 409

13.5. Classes autorisées dans la zone internet 409

13.6. Liste des touches de raccourcis pour les commandes d'édition 411

13.7. Liste des classes par catégories 413

13.8. Liste des couleurs prédéfinies 415

14 Index 421

Introduction

Avertissement 12
Prérequis 12
Présentation de XAML 13
Utiliser XAMLPad 17
Checklist 19

1.1 Avertissement

Ce livre vous fera découvrir progressivement XAML au moyen d'exemples. Nous partirons des fonctionnalités les plus simples et les plus utiles pour aller vers des notions un peu plus complexes. Toutefois, afin que vous puissiez facilement revenir par la suite sur une fonctionnalité particulière, les éléments sont regroupés le plus possible dans des chapitres dédiés. À la fin du livre, vous trouverez un récapitulatif des classes et des attributs utilisés. Celui-ci se veut non pas un référentiel complet mais plutôt un récapitulatif des notions vues.

Il s'agit d'une présentation du XAML et non de WinFX. C'est pourquoi nous nous attacherons principalement aux techniques accessibles depuis XAML, même si à certains moments nous devrons évidemment voir ou réaliser le code .NET associé.

Au moment où ces lignes ont été écrites, le code XAML et la bibliothèque WinFX avec laquelle il travaille étaient toujours en version bêta et susceptibles d'être modifiés. Il se peut que certaines fonctionnalités aient entre-temps été ajoutées ou retirées. Comme nous nous intéresserons surtout aux fonctions les plus courantes, il est probable que le contenu de ce livre respecte les règles en vigueur dans la version définitive du produit.

1.2 Prérequis

Avant de pouvoir développer avec XAML et WinFX, vous devez au préalable installer le Framework .NET 2.0 ainsi qu'une version de Visual Studio 2005.

Vous pourriez utiliser n'importe quel éditeur de texte, et ce y compris Notepad, mais l'utilisation de Visual Studio vous facilitera grandement la vie, que ce soit pour la création des projets ou la compilation de votre application. Il vous apporte également l'IntelliSense, ce qui est un très gros avantage.

Vous devez ensuite installer le kit de développement WinFX. Attention, Windows XP SP2, Windows 2003 ou Windows Vista est requis pour pouvoir l'installer !

Pour réaliser les exemples, nous utiliserons d'une part "XAMLPad", un petit utilitaire livré avec le kit de développement et d'autre part "Microsoft Visual Basic 2005 Express Edition".

Afin de bénéficier de toutes les fonctionnalités dans Visual Studio, vous devez également installer l'extension pour Visual Studio.

L'ensemble de ces éléments sera repris dans le Framework 3.0, dont la sortie est attendue pour le début de l'année 2007.

1.3 Présentation de XAML

Qu'est-ce que XAML ?

Le XAML est un des composants de la nouvelle technologie *Windows Presentation Fundation* (en abrégé, WPF). Cette technologie est accessible *via* le kit de développement WinFX.

WinFX doit remplacer l'ancien API Win32. Évidemment, pour des raisons de portabilité, l'ancien API sera encore présent dans les futures versions de Windows, mais jusqu'à quand ? Il a initialement été créé spécifiquement pour Windows Vista, mais Microsoft a décidé de le rendre disponible pour Windows XP (SP2) et Windows 2003. WinFX est intimement lié au Framework .NET 2.0. L'avantage d'utiliser WinFX par rapport à Win32 est qu'il apporte une approche vectorielle de l'affichage et offre des possibilités 3D.

WinFX sera en définitive inclus dans le Framework 3.0, qui contiendra :

- Windows Communication Fundation ;
- Windows Presentation Fundation ;
- Windows Workflow Fundation ;
- Windows CardSpace ;
- .NET Framework 2.0.

XAML, quant à lui, est un langage de description fondé sur la norme XML. Contrairement au XML, les noms des balises et leur contenu doivent respecter une syntaxe stricte et correspondre à une classe de WPF. Tout ce qui est fait en XAML peut également être fait dans du code traditionnel. De même, XAML ne supporte qu'une partie du modèle offert par WPF. Il offre en revanche une beaucoup plus grande lisibilité et une séparation entre le code logique et l'interface graphique. Le XAML va nous permettre de décrire les écrans de l'application. Le reste du traitement se fait de manière traditionnelle, *via* du code .NET.

Dans le cadre de ce guide d'initiation, les parties .NET des exemples sont en Visual Basic .NET mais, si vous préférez, vous pouvez bien entendu utiliser du C# ou n'importe quel autre langage .NET. L'apprentissage du .NET n'est pas l'objectif de ce guide ; néanmoins, pour une bonne compréhension des exemples, il sera souvent nécessaire de voir aussi bien la partie XAML que la partie .NET.

Afin de mieux comprendre la place de WinFX, de .NET et de XAML dans l'architecture, vous pouvez vous référer au schéma ci-dessous.

Comme vous pouvez le constater, XAML, comme les langages .NET, forme la couche supérieure et représente l'interface avec le développeur. En dessous, nous retrouvons les couches composées de code managé ainsi que la CLR nécessaire pour l'exécution de ce code. La dernière partie est la couche de communication avec Windows lui-même. À noter que WPF est pour sa part composé effectivement de Presentation Framework, de Presentation Core et de Milcore.

Petits rappels XML

Vous n'avez pas à connaître XML pour utiliser XAML. Les quelques notions nécessaires à la bonne compréhension vont vous être expliquées dans ce chapitre.

Le XML présente les informations de façon structurée et hiérarchique sous forme de texte.

Un fichier XML contient un ou des nœuds dans lesquels s'imbriquent d'autres nœuds appelés nœuds enfants.

Mais qu'est-ce qu'un nœud ?

```
<NomDuNoeud>
      <NoeudEnfant>
         Valeur
      </NoeudEnfant>
      <NoeudEnfant>
         Valeur
      </NoeudEnfant>
</NomDuNoeud>
```

Un nœud est tout ce qui se trouve entre une balise ouvrante <NomDuNoeud > et une balise fermante </ NomDuNoeud>.

Les attributs sont des informations qui se placent dans la balise ouvrante.

```
<NomDuNoeud MonAttribut= "valeur">
```

L'attribut est toujours suivi du symbole = et d'une valeur entre guillemets. Si le nœud ne contient que des attributs, la syntaxe autorise de ne pas placer de balise fermante. La balise ouvrante se termine alors par le symbole /.

```
<NomDuNoeud MonAttribut= "valeur" />
```

Pour commenter du code XML, vous inclurez les commentaires dans une balise spécifique.

```
<!-- votre commentaire -->
```

Le commentaire peut sans problème s'étendre sur plusieurs lignes.

Les principes généraux

Le principe général est finalement assez simple, chaque balise XAML identifie une instance d'une classe de la librairie WinFX. L'instanciation de cet objet est effectuée automatiquement. Vous ne devez ni le déclarer ni l'instancier avec `New` dans votre code .NET.

Par exemple,

```
<Window />
```

permet d'instancier un objet de la classe `Window`. Si vous désirez initialiser une propriété de la classe, il suffit d'ajouter un attribut dans le nœud XML ou, selon les cas, de créer un nœud enfant.

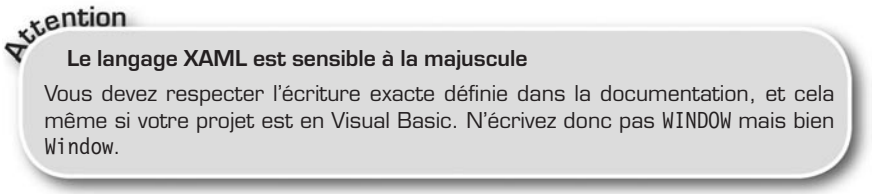
Exemple :

```
<Window Title= "Titre" />
```

Cette balise permet d'assigner la valeur `"Titre"` à la propriété `Titel` de la classe `Window`. Notez que même les propriétés numériques doivent être assignées comme une chaîne de caractères. Pour les valeurs booléennes, vous devez assigner `True` ou `False` entre guillemets.

Pour les collections, il s'agit de définir des nœuds enfants. Par exemple :

```
<Window>
   <Window.InputBindings>
   </Window.InputBindings>
< /Window>
```



Comme en XAML les noms des attributs des balises XML correspondent aux noms des propriétés de la classe associée, nous parlerons indistinctement dans ce livre d'attribut ou de propriété.

Pour mieux comprendre cette notion de correspondance entre XAML et les langages .NET, sachez que les deux bouts de code ci-dessous ont la même fonction.

En XAML :

```
<Label Name="lblNom" Width="60" Height="20">
   mon nom
</Label>
```

En VB .NET :

```
Dim lblNom as new Label
lblNom.Width = 60
lblNom.Height = 20
lblNom.Text = "mon nom"
```

Comme vous pouvez le constater, ce qui est fait en XAML peut parfaitement être fait en code .NET. Toutefois, la bonne pratique veut que l'on utilise XAML pour décrire l'interface et les langages .NET traditionnels pour les traitements. Pour une même classe, vous aurez alors vraisemblablement deux fichiers de sources, un en XAML et l'autre en code .NET. Les deux seront fusionnés lors de la compilation.

Nous reviendrons sur les différents fichiers automatiquement générés ou non au cours de ce livre, au fur et à mesure des besoins et de l'explication des différentes notions. Sachez toutefois que du code XAML seul peut déjà être compris par le système et interprété sans même devoir utiliser un compilateur. C'est par ce genre de code que nous allons commencer.

XAML contient malgré tout quelques petites subtilités que nous allons détailler.

Tout d'abord, le code XAML doit contenir un élément *root* comme le veut XML, mais cet élément doit être un élément faisant partie du modèle d'application comme `Window` ou `Page` ou encore un conteneur tel `Grid` ou `StackPanel`. Nous reviendrons ultérieurement sur la notion de conteneur et d'éléments d'application. Cet élément *root* doit contenir comme attribut la déclaration de deux *Namespaces* au minimum. Cette déclaration se fait en utilisant l'attribut classique `xmlns`.

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
```

S'il est nécessaire de faire référence à d'autres *namespaces*, il sera nécessaire d'ajouter d'autres attributs `xmlns` en respectant la syntaxe ci-dessous.

```
xmlns:nom="clr-namespace:nom du namespace;assembly=nom de l'assembly"
```

L'*assembly* peut être omis si le *namespace* est contenu dans le même *assembly* que le programme.

XAML autorise également l'utilisation d'attributs particuliers, dont voici les principaux.

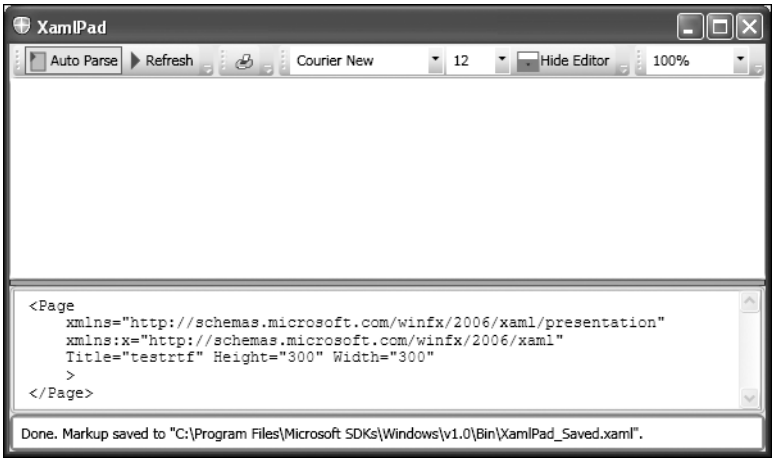
Attributs spécifiques	
Attribut	**Utilité**
`x:Name`	Cet attribut a la même fonction que l'attribut `Name`, qui est généralement présent. Ils sont totalement interchangeables mais ne peuvent être utilisés simultanément. Préférez `Name` à `x:Name` et limitez l'usage de ce dernier aux objets qui ne disposent pas de la propriété `Name`.
`x:Key`	Est encore un moyen de donner un nom mais, typiquement, ce type de nommage est utilisé pour nommer des ressources. Il ne s'agit en aucun cas du même usage que `x:Name`.
`x:Class`	Attribut placé dans l'élément `root` ; il fait le lien avec la classe définie dans le *code behind* (.NET).
`x:ClassModifier`	Permet de spécifier d'autres modificateurs que `partial` et `public`, qui sont les valeurs par défaut.
`x:Null`	Pour définir une valeur nulle à une propriété.
`x:Code`	Permet d'introduire du code .NET dans le fichier XAML. Il sera généralement utilisé en conjonction avec `CDATA` : `<x:Code> <![CDATA[ Code … ]]> </x:Code>`
`x:Static`	Pour accéder à une valeur statique d'une classe.
`x:Type`	Dans XAML, il a le même effet que `TypeOf` dans le code .NET.
`x:TypeArgument`	Pour déterminer le type des arguments qui doivent être reçus. Cet attribut est principalement utilisé par `PageFunction`.
`x:Array`	Permet d'instancier un `array` comme valeur d'un attribut. Toutefois, il n'est pas possible de remplir cet `array` dans le code XAML.

Ces différentes notions seront abordées par l'exemple dans la suite de ce livre.

1.4 Utiliser XAMLPad

L'utilitaire XAMLPad est très simple d'emploi et permet de visualiser très rapidement du code XAML. Il est toutefois limité au XAML statique. C'est pour cette raison que certains exemples seront réalisés avec Visual Studio.

XamlPad se présente comme une fenêtre en deux parties. La partie supérieure affiche le résultat du code et la partie inférieure, le code lui-même.



▲ **Figure 1-1 :** L'outil XAMLPad livré avec le kit de développement

En résumé, vous tapez le code en bas, il affiche le résultat en haut. C'est aussi simple que cela.

Pour faciliter la vision, vous disposez de quelques options :

- *Auto Parse*, qui est activée par défaut et traite en temps réel votre code XAML. Ainsi, vous constatez directement l'impact de vos changements.

Astuce

Long code source

Si votre code est long, vous pouvez désactiver l'option AutoParse. Cela vous évitera de désagréables effets de rafraîchissement.

- *Refresh*. N'est vraiment utile que si l'*Auto Parse* est désactivée.
- L'icône d'imprimante, qui imprime le résultat mais pas le code.

Avec les autres options de la barre d'outils, vous pouvez changer la police utilisée pour le code, choisir de cacher temporairement le code ou encore faire un zoom de la fenêtre d'affichage du résultat.

Consultez la barre de statut, elle vous donne de précieux renseignements sur votre code en cours de frappe.

Il est dommage que l'IntelliSense ne soit pas disponible dans cet utilitaire. Si vous désirez en disposer, vous pouvez reproduire les exemples dans Visual Studio. Mais vous devrez alors préalablement créer un projet spécifique et demander chaque fois l'exécution pour visualiser le résultat.

Vous pouvez vous reporter au chapitre Créer une application Windows *page 132 pour vous aider à créer votre projet.*

1.5 Checklist

Les notions essentielles que nous avons vues dans ce premier chapitre sont :

- le matériel nécessaire pour débuter en programmation XAML ;
- les bases du XML ;
- comment fonctionne XAML ;
- comment utiliser l'outil XAMLPad.

Fonctionnalités de base

Afficher du texte 22
Introduire du texte 38
Créer un bouton 46
Afficher un cadre 47
Afficher une image 48
Checklist 53

Dans ce chapitre, nous allons voir de manière individuelle les différents éléments de base nécessaires à la réalisation d'une interface utilisateur. Cela nous permettra d'aborder l'affichage et la saisie de texte, l'affichage d'une image et l'utilisation d'un bouton. Avec ce minimum de connaissance, nous pourrons ensuite passer à l'étape suivante, qui sera le placement des éléments. C'est seulement après que nous aborderons les autres contrôles classiquement utilisés. Dans le but de grouper les fonctionnalités, nous ne nous contenterons pas de survoler ces fonctionnalités mais les verrons assez en profondeur.

2.1 Afficher du texte

XAML offre deux possibilités de base pour l'affichage de texte. Dans ce chapitre, nous verrons les deux méthodes et découvrirons les avantages de l'une et de l'autre. Nous découvrirons en même temps de nombreuses possibilités qui pourront être exploitées avec la majorité des contrôles.

Le premier des contrôles est de loin le plus connu puisqu'il s'agit du `Label`, l'autre est le `TextBlock`, que nous allons découvrir ensemble.

Avec un Label

Le `Label` est probablement le contrôle le plus utilisé. Il reste très simple à manipuler mais, comme vous allez le constater, il nous réserve d'agréables surprises.

Pour créer un label, vous devez utiliser la balise du même nom.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <StackPanel>
      <Label>
         Mon premier label.
      </Label>
   </StackPanel>
</Page>
```

Ne vous souciez pas pour l'instant des balises `Page` et `StackPanel`. Nous verrons ultérieurement à quoi elles servent et comment les utiliser. Sachez seulement que, sans la balise `Page` ou une autre balise racine valide, vous ne pourrez visualiser le résultat. La balise `StackPanel` sert de conteneur.

*Nous reviendrons sur les conteneurs dans le chapitre **Disposer les éléments à l'écran** sur la mise en page (page 56).*

Dans la suite, ces balises ne seront plus systématiquement reprises au sein des exemples mais elles devront évidemment être présentes.

◄ **Figure 2-1** : Une simple étiquette

Au lieu de placer le code entre la balise de début et la balise de fin du nœud `Label`, vous pouvez également opter pour assigner l'attribut `Content`. Le résultat sera le même.

La syntaxe devient alors :

```
<Label Content="Mon premier label."/>
```

Remarque

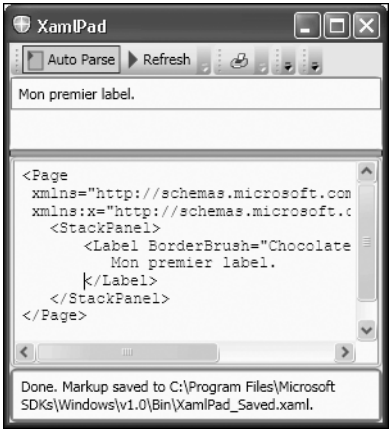
Portée des explications

Les attributs que nous allons voir dans cette partie seront également utilisables pour les autres contrôles que nous aborderons ultérieurement.

Comme beaucoup de contrôles, le `Label` occupe un espace rectangulaire. Nous pouvons nous en rendre compte en affichant un bord autour de notre contrôle.

```
<Label BorderBrush="Chocolate" BorderThickness="1">
   Mon premier label.
</Label>
```

Les deux attributs sont nécessaires car aucune couleur n'est définie par défaut et la taille est de zéro.



◄ **Figure 2-2** : Une étiquette dans un cadre

La taille occupée dépend de divers paramètres comme la taille de la fenêtre.

Si vous souhaitez contrôler la taille de cette zone, vous pouvez utiliser les attributs `Width` et `Height`. Dans l'exemple, nous allons fixer la taille à 200 pixels de large sur 60 pixels de haut.

```
<Label BorderBrush="Chocolate" BorderThickness="1"
 Width="200" Height="60">
   Mon premier label.
</Label>
```

Comme vous pouvez le constater, la zone occupée a été modifiée.

◄ **Figure 2-3** : La taille d'une étiquette

Remarque

Centrage

Notez au passage que par défaut le contrôle est centré horizontalement.

Ce n'est pas le seul moyen de contrôler la taille du `Label`. XAML met également à notre disposition les attributs `MinWidth`, `MinHeight`, `MaxWidth` et `MaxHeight`.

Avec ces attributs, vous laissez la taille s'adapter à l'environnement tout en fixant des valeurs limites.

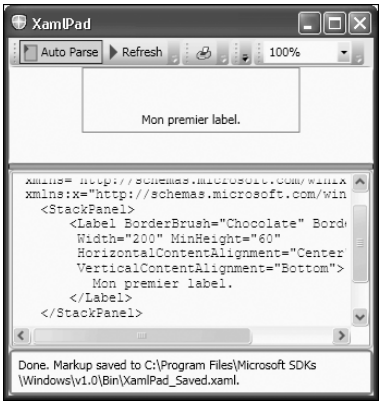
```
<Label BorderBrush="Chocolate" BorderThickness="1"
 MinWidth="100" MinHeight="30"
 MaxWidth="300" MaxHeight="80">
   Mon premier label.
</Label>
```

◄ **Figure 2-4 :** Une étiquette avec MinWidth et MaxWidth

Si la position du texte ne vous convient pas, vous avez l'opportunité de modifier l'alignement horizontal et vertical. Vous pouvez utiliser les attributs `HorizontalContentAlignment` et `VerticalContentAlignment`.

```
<Label BorderBrush="Chocolate" BorderThickness="1"
 Width="200" MinHeight="60"
 HorizontalContentAlignment="Center"
 VerticalContentAlignment="Bottom">
   Mon premier label.
</Label>
```

Les valeurs possibles sont respectivement `Left`, `Rigth`, `Center`, `Stretch` et `Top` pour l'alignement horizontal et `Bottom`, `Center` et `Stretch` pour l'alignement vertical.



◄ **Figure 2-5 :** Alignement du contenu d'une étiquette

Attention

Ne pas confondre

Ne confondez pas ces deux attributs avec les attributs `VerticalAlignment` et `HorizontalAlignment`, qui permettent de centrer le contrôle dans son conteneur.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Label BorderBrush="Chocolate" BorderThickness="1"
    HorizontalAlignment="Center"
    VerticalAlignment="Center">
      Mon premier label.
   </Label>
</Page>
```

Remarque

Absence de la balise StackPanel

Pour une meilleure compréhension, dans cet exemple, la balise `StackPanel` a été retirée. C'est pourquoi le code est présenté complet.

◄ **Figure 2-6 :** Centrer l'étiquette

Si la police par défaut ne vous convient pas, vous disposez de cinq attributs différents pour l'adapter à votre goût.

L'attribut `FontFamily` permet de sélectionner le type de police comme Arial ou Verdana. L'attribut `FontSize` détermine la taille.

```
<Label HorizontalAlignment="Center"
 VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14"
 FontWeight="Heavy" FontStyle="Italic"
 FontStretch="UltraExpanded">
   Mon premier label.
</Label>
```

◄ **Figure 2-7 :** Affichage dans une police différente

Contrairement à ce que nous connaissions avant, les possibilités sont bien plus étendues. `FontStrech` accepte pas moins de dix valeurs différentes allant de `UltraCondensed` à `UltraExpanded` alors que `FontWeight` accepte quatorze valeurs allant de `Thin` à `Heavy`. La taille du font peut varier de 1 à 35 791.

En ce qui concerne les couleurs, nous disposons des attributs très traditionnels `Foreground` et `Background`.

```
<Label  HorizontalAlignment="Center"
 VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14"
 FontWeight="Heavy" FontStyle="Italic"
 FontStretch="UltraExpanded"
 Foreground="Aquamarine" Background="Blue">
   Mon premier label.
</Label>
```

◄ **Figure 2-8 :** Couleur de fond et d'avant-plan

Nous avons vu les principales possibilités offertes avec le contrôle `Label` et qui sont par ailleurs applicables aux autres contrôles classiques que nous verrons dans ce chapitre.

Attention

Nommer ses contrôles

Pour pouvoir ultérieurement interagir avec les différents contrôles que vous avez créés, ils doivent porter un nom. Bien que vous n'ayez pas à interagir avec tous les contrôles, je vous conseille de les nommer tous, et ce y compris les `Label`. Votre code s'en trouvera beaucoup plus clair.

Pour nommer le contrôle, vous devez utiliser l'attribut `Name`. Vous vous rendrez vite compte qu'il est utile d'utiliser systématiquement cet attribut.

```
<Label Name="lblPremierLabel"
 HorizontalAlignment="Center"
 VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14"
 FontWeight="Heavy" FontStyle="Italic"
 FontStretch="UltraExpanded"
 Foreground="Aquamarine" Background="Blue">
   Mon premier label.
</Label>
```

Astuce

Donner toujours des noms clairs à ses contrôles

La compréhension finale de votre programme sera grandement accrue si les noms donnés sont clairs et sans ambiguïté. Si, sur le moment, cela vous paraît superflu, lors de modifications ultérieures vous serez très heureux de l'avoir fait. Dans les exemples, en plus d'un nom explicite, je commence par un préfixe qui identifie le type de contrôle. Il existe d'autres conventions de nommage, chacune ayant ses avantages et ses inconvénients. Choisissez-en une et tenez-vous-y.

Vous pouvez par exemple opter pour cette convention de nommage.

Exemple de convention de nommage

Type	Convention
Nom des classes	`CamelCase`. Ex. : `MaClasse` Remarque : `CamelCase` est un terme courant qui signifie que chaque mot dans le nom donné commence par une majuscule et que le reste du mot est en minuscule.
Nom d'un membre public	`CamelCase` suivi de `_m`. La bonne pratique veut que les membres publics soient évités. Préférez un membre privé associé à une propriété pour y accéder. Ex. : `NomDeRue_m`
Nom d'un membre privé	`CamelCase` commençant par une minuscule et terminé par `_m`. Ex. : `nomDeRue_m`
Nom d'une propriété publique	`CamelCase` commençant par une majuscule. Si la propriété est associée à un membre, le nom de la propriété sera le même que le nom du membre associé mais sans le `_m`. Ex. : `NomDeRue`

Exemple de convention de nommage	
Type	**Convention**
Nom d'une propriété privée	`camelCase` commençant par une minuscule Si la propriété est associée à un membre, le nom de la propriété sera le même que le nom du membre associé mais sans le `_m`. Ex. : `nomDeRue`
Nom de variable	`CamelCase` commençant par une minuscule. Ex. : `nomDeRue` (en VB, vu que le langage est insensible aux majuscules, il sera probablement utile d'ajouter le suffixe `_v` pour éviter la confusion avec la propriété)
Nom d'un contrôle	`CamelCase` terminé par le type de contrôle. Microsoft préconise maintenant que le type soit écrit dans son entièreté (`TextBox` et non `Txt`). Ex. : `nomDeRueTextBox` Comme d'ailleurs dans ce livre, vous trouverez souvent le type de contrôle en préfixe et non en suffixe. Toutefois, le fait de placer le type à la fin a l'avantage de regrouper les éléments concernant une même donnée dans l'IntelliSense. La bonne pratique veut que les contrôles soient privés. Si tel n'est pas le cas, le nom doit commencer par une majuscule.
Constante	Tout en majuscule précédé de `_`. Ex. : `_CHEMINDUFICHIER`
Espace de nom	`CamelCase`. Ex. : `MonProgramme.Operation`
Énumération	`CamelCase` Ex. : `Couleur.Bleu  Couleur.Vert`

Avec un TextBlock

L'origine du `TextBlock` est plutôt Web que Windows. Il fait pourtant son entrée dans l'API disponible pour les applications Windows classiques.

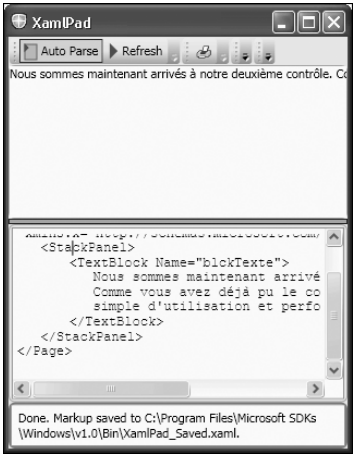
Contrairement au contrôle de type `Label`, `TextBlock` n'hérite pas de la classe `Control`. `TextBlock` est optimisé pour l'affichage de longs textes.

Je ne reviendrai pas sur la manière de modifier les couleurs ou la police, il suffit d'utiliser à l'identique les techniques vues précédemment.

Le `TextBlock` nécessite un peu plus de mise en forme que le `Label`, qui se contente d'un minimum. En effet, si nous utilisons la commande ci-dessous :

```
<TextBlock Name="blckTexte">
   Nous sommes maintenant arrivés à notre deuxième contrôle.
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
</TextBlock>
```

le résultat ne sera pas à la hauteur de nos espérances.



◄ **Figure 2-9 :** Un bloc de texte

Il est possible de déterminer une taille à la zone d'affichage, et ainsi nous pouvons demander un arrangement automatique du texte.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70" TextWrapping="Wrap">
   Nous sommes maintenant arrivés à notre deuxième contrôle
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
</TextBlock>
```

◄ **Figure 2-10 :** Un bloc de texte multiligne

Ce qui est beaucoup plus conforme au résultat attendu.

Astuce

Limiter la taille au lieu de la fixer

Au lieu de fixer la taille, choisissez de délimiter une taille maximale. De cette manière, si nous réduisons la taille de la fenêtre, le texte s'adapte au mieux aux modifications.

◄ **Figure 2-11** : Le même bloc

L'attribut `TextWrapping` peut prendre les valeurs `NoWrap`, `Wrap` mais également `WrapWithOverflow`. Dans ce dernier cas, la taille du contrôle sera automatiquement agrandie si nécessaire, même si vous avez défini une taille avec l'attribut `Height`.

Bien sûr, si pour quelque motif que ce soit la zone réservée au `TextBlock` devient trop petite, le texte ne pourra plus être entièrement affiché. Dans ce cas, l'attribut `TextTrimming` nous permettra d'inclure automatiquement les « … » signalant ainsi à l'utilisateur que le texte est incomplet.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70"
 TextWrapping="Wrap" TextTrimming="WordEllipsis">
   Nous sommes maintenant arrivés à notre deuxième contrôle
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
</TextBlock>
```

◄ **Figure 2-12** : Un bloc de texte avec coupure

L'attribut TextTrimming, en plus de la valeur par défaut (None), peut prendre deux valeurs différentes. Soit WordEllipsis, comme dans notre exemple, où l'arrêt se fait après un mot entier, soit CharacterEllipsis, qui provoque l'arrêt derrière n'importe quel caractère.

Comme vous pouvez le constater, le contrôle se place très près des bords de son conteneur, ce qui n'est pas forcément du meilleur effet. Vous pouvez définir une marge tout autour de votre contrôle en utilisant l'attribut Margin. Cet attribut n'est pas spécifique au TextBlock et nous aurions déjà pu l'utiliser avec le contrôle de type Label.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70"
 TextWrapping="Wrap" TextTrimming="WordEllipsis"
 Margin="5,5,5,5" >
   Nous sommes maintenant arrivés à notre deuxième contrôle.
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
</TextBlock>
```

◄ **Figure 2-13** : Marges autour d'un contrôle

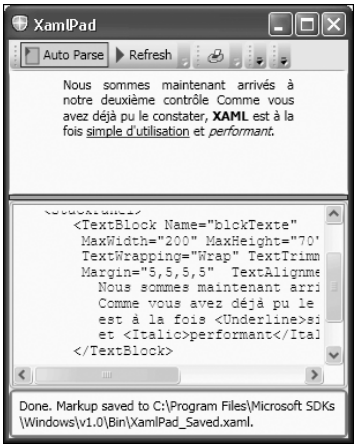
Pour terminer les spécificités du TextBlock par rapport au Label, il nous reste à voir l'attribut TextAlignment. Les valeurs possibles sont les très classiques Left, Right, Center et Justify.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70"
 TextWrapping="Wrap" TextTrimming="WordEllipsis"
 Margin="5,5,5,5" TextAlignment="Justify" >
   Nous sommes maintenant arrivés à notre deuxième contrôle.
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
</TextBlock>
```

◄ **Figure 2-14 :** Alignement du texte

De plus, le `TextBlock` permet d'enrichir le contenu. Vous pouvez par exemple y inscrire du texte en gras, en italique ou souligné.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70"
 TextWrapping="Wrap" TextTrimming="WordEllipsis"
 Margin="5,5,5,5" TextAlignment="Justify" >
   Nous sommes maintenant arrivés à notre deuxième contrôle
   Comme vous avez déjà pu le constater, <Bold>XAML</Bold>
   est à la fois <Underline>simple d'utilisation</Underline>
   et <Italic>performant</Italic>.
</TextBlock>
```



◄ **Figure 2-15 :** Enrichissement du texte

Si ces possibilités d'enrichissement ne vous suffisent pas, vous pouvez utiliser `TextDecoration`. Cette propriété n'est pas spécifique à `TextBlock`, et vous pouvez la retrouver quasiment partout où du texte est affiché. Elle est accessible grâce à la balise `TextBlock`. `TextDecorations`, pour un autre contrôle, remplace `TextBlock` par la valeur appropriée. Elle permet d'ajouter des décorations à vos caractères. La description de la décoration est définie à l'intérieur du nœud `TextDecoration`. La décoration est en réalité un trait qui peut être placé à quatre endroits différents. La position est définie au moyen de la propriété `Location`. Elle peut être au-dessus du caractère, au milieu du caractère, sous le caractère ou juste sous le caractère. La taille, la couleur, la forme du trait sont définies avec un objet de type `Pen`.

Ci-dessous, vous trouverez un code complet reprenant les différentes possibilités énumérées ci-avant.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel>
      <TextBlock TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" Width="800"
       HorizontalAlignment="Center">
         <Bold>Non seulement vous pouvez utiliser
         l'enrichissement classique</Bold>
         mais vous pouvez aussi utiliser un enrichissement
         plus complexe
         <LineBreak/>
```

Le `TextBlock` suivant utilise une décoration au milieu du caractère. Le trait est rouge et d'une épaisseur de 2 points.

```
         <TextBlock>
            <TextBlock.TextDecorations>
               <TextDecoration Location="Strikethrough">
                  <TextDecoration.Pen>
                     <Pen Brush="Red" Thickness="2"/>
                  </TextDecoration.Pen>
               </TextDecoration>
            </TextBlock.TextDecorations>
            comme du barré coloré
         </TextBlock>
         <LineBreak/>
```

Le `TextBlock` suivant utilise une décoration sous le caractère. Le trait est bleu et toujours d'une épaisseur de 2 points.

```
<TextBlock>
   <TextBlock.TextDecorations>
      <TextDecoration Location="Underline">
         <TextDecoration.Pen>
            <Pen Brush="Blue" Thickness="2"/>
         </TextDecoration.Pen>
      </TextDecoration>
   </TextBlock.TextDecorations>
   comme un soulignement en bleu
</TextBlock>
<LineBreak/>
```

Le `TextBlock` suivant utilise une décoration juste sous le caractère. Le trait est noir et d'une épaisseur de 1 point.

```
<TextBlock>
   <TextBlock.TextDecorations>
      <TextDecoration Location="Baseline">
         <TextDecoration.Pen>
            <Pen Brush="Black" Thickness="1"/>
         </TextDecoration.Pen>
      </TextDecoration>
   </TextBlock.TextDecorations>
   comme une fine ligne sous les caractères
</TextBlock>
<LineBreak/>
```

Le `TextBlock` suivant utilise une décoration au-dessus du caractère. Le trait est rouge et d'une épaisseur de 2 points.

```
<TextBlock>
   <TextBlock.TextDecorations>
      <TextDecoration Location="OverLine">
         <TextDecoration.Pen>
            <Pen Brush="Red" Thickness="2"/>
         </TextDecoration.Pen>
      </TextDecoration>
   </TextBlock.TextDecorations>
   ou encore une ligne au dessus
</TextBlock>
<LineBreak/>
```

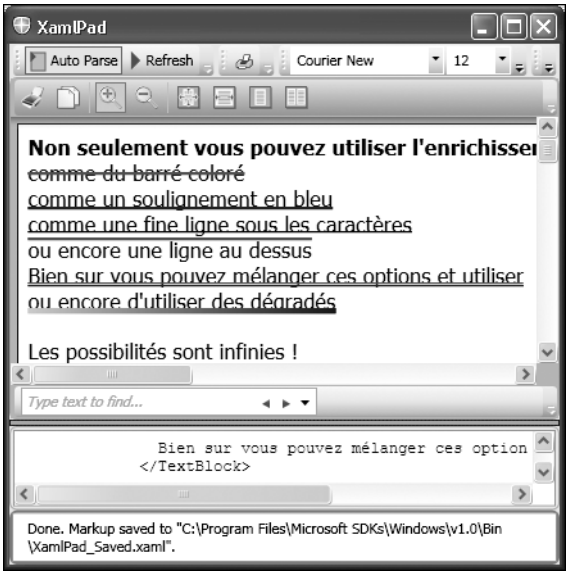
Le `TextBlock` suivant utilise plusieurs décorations. Une première juste sous le caractère. Le trait est rouge, d'une épaisseur de 1 point et discontinu. La seconde est sous le caractère. Le trait est bleu, d'une épaisseur de 1 point et est continu. Les deux ensembles réalisent un double trait spécifique.

```
<TextBlock>
   <TextBlock.TextDecorations>
      <TextDecoration Location="Baseline">
         <TextDecoration.Pen>
            <Pen Brush ="Red" Thickness="1">
               <Pen.DashStyle>
                  <DashStyle Dashes="1"/>
               </Pen.DashStyle>
            </Pen>
         </TextDecoration.Pen>
      </TextDecoration>
      <TextDecoration Location="Underline">
         <TextDecoration.Pen>
            <Pen Brush="Blue" Thickness="1"/>
         </TextDecoration.Pen>
      </TextDecoration>
   </TextBlock.TextDecorations>
   Bien sur vous pouvez mélanger ces options
   et utiliser
</TextBlock>
<LineBreak/>
```

Ce dernier `TextBlock` utilise une décoration sous le caractère. Le trait est d'une épaisseur de 4 points et d'un dégradé allant d'un bleu léger à un bleu foncé.

```
<TextBlock>
   <TextBlock.TextDecorations>
      <TextDecoration Location="Underline">
         <TextDecoration.Pen>
            <Pen Thickness="4">
               <Pen.Brush>
                  <LinearGradientBrush>
                     <LinearGradientBrush
                      .GradientStops>
                       <GradientStop
                        Color="LightBlue"
                        Offset="0"/>
                       <GradientStop
                        Color="DarkBlue"
                        Offset="1"/>
                     </LinearGradientBrush
                      .GradientStops>
                  </LinearGradientBrush>
               </Pen.Brush>
            </Pen>
         </TextDecoration.Pen>
      </TextDecoration>
   </TextBlock.TextDecorations>
   ou encore d'utiliser des dégradés
```

```
        </TextBlock>
        <LineBreak/>
        <LineBreak/>
        Les possibilités sont infinies !
      </TextBlock>
   </StackPanel>
</Page>
```



◄ **Figure 2-16 :** Utiliser des décorations

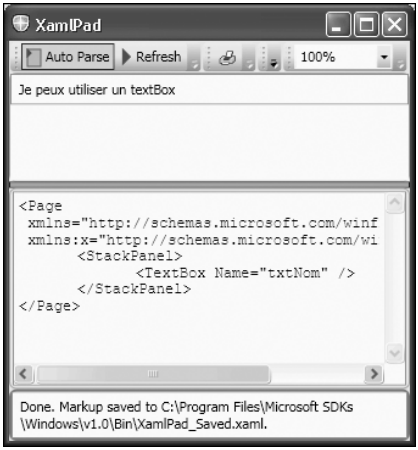
Notez au passage l'utilisation de la balise `LineBreak` pour provoquer un passage à ligne imposé.

Dans l'exemple, vous aurez pu constater la présence d'un dégradé ; si vous souhaitez en savoir plus sur cette fonctionnalité, elle est traitée au chapitre Créer un style page 229.

2.2 Introduire du texte

Le contrôle `TextBox` est, avec le `Label`, probablement le plus utilisé. Il est aussi un des plus simples. Le code suivant va déjà nous permettre d'obtenir un `TextBox` tout à fait fonctionnel.

```
<TextBox Name="txtNom" />
```



◄ **Figure 2-17 :** Une boîte de texte

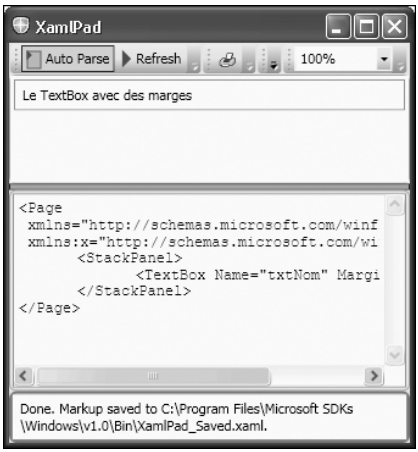
Nous pouvons appliquer à ce contrôle tout ce que nous avons déjà vu pour le contrôle `Label`.

Astuce

Appliquer des marges

Pour une question de lisibilité et de mise en page, je vous conseille d'appliquer systématiquement des marges à vos contrôles de type `TextBox`.

```
<TextBox Name="txtNom" Margin="3,3,3,3" />
```

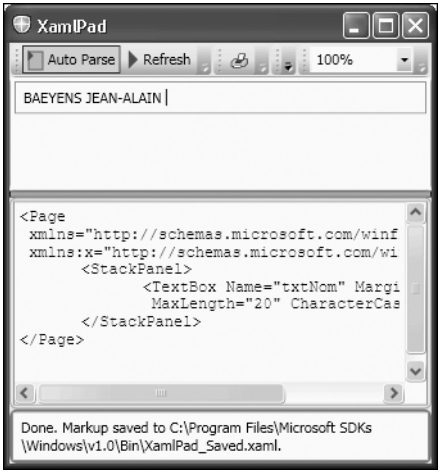


◄ **Figure 2-18 :** Un TextBox avec marges

L'attribut `MaxLength` va nous permettre de limiter l'encodage. Ce qui est bien pratique quand le contenu doit être enregistré dans une base de données. Vous évitez ainsi une perte d'information à l'insu de l'utilisateur ou des problèmes plus tard dans le code.

L'attribut `CharacterCasing` limite lui aussi l'encodage en imposant automatiquement le texte en majuscule (valeur `Upper`) ou en minuscule (valeur `Lower`). J'aurais également apprécié la possibilité de forcer la première lettre en majuscule et les autres en minuscules mais, actuellement, ce n'est pas prévu.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 MaxLength="20" CharacterCasing="Upper" />
```



◄ **Figure 2-19 :** Majuscules imposées

Un `TextBox` peut également être multiligne. Dans ce cas, vous aurez la possibilité d'ajuster son utilisation avec les attributs `AcceptsReturn`, `AcceptsTab` et `TextWrapping`.

L'attribut `AcceptsReturn` autorise ou non l'utilisation du passage à la ligne imposé (touche Entrée). L'attribut `AcceptsTab` autorise ou non l'emploi de la tabulation.

Remarque

Déplacement par tabulations

Si vous autorisez l'utilisation de la tabulation, elle ne pourra évidemment plus être utilisée pour passer au champ suivant. Toutefois, mais il faut le savoir, la combinaison Ctrl+Tab remplit aussi cette fonction.

L'attribut `TextWrapping` détermine le comportement du contrôle quand le texte frappé arrive en bout de ligne. Les valeurs possibles sont `NoWrap`, `Wrap` et `WrapWithOverflow`. Cette dernière va non seulement provoquer un passage à la ligne automatique mais, contrairement à `Wrap`, également étendre vers le bas la zone prévue initialement.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 MinHeight="80" AcceptsReturn="True"
 AcceptsTab="True" TextWrapping="WrapWithOverflow" />
```

◄ **Figure 2-20 :** Tabulation dans un TextBox

Dans l'exemple ci-dessus, le texte arrive sur la dernière ligne. Si d'aventure nous continuons la frappe, vous pouvez constater l'effet de `WrapWithOverflow`.

◄ **Figure 2-21 :** Zone de débordement

Astuce

Combiner la zone de débordement et la hauteur maximale

Si vous utilisez l'option `WrapWithOverflow`, il est préférable de fixer une taille maximale en utilisant l'attribut `MaxHeight`.

Remarque

Gestion des lignes

Comme vous pouvez le remarquer, rien n'indique qu'il s'agisse ou non d'un `TextBox` multiligne. En fait, un `TextBox` a toujours les capacités multiligne mais, si vous n'autorisez ni le passage automatique à la ligne (*Wrap*) ni le retour à la ligne imposé, l'encodeur ne pourra pas créer de seconde ligne. Le `TextBox` est capable de faire défiler le texte aussi bien horizontalement que verticalement quand cela est nécessaire.

Les attributs `MinLines` et `MaxLines` sont une bonne alternative à l'utilisation des attributs `MinHeight` et `MaxHeight`. `MinLines` et `MaxLines` fixent respectivement le nombre minimal et maximal de lignes visibles. La taille sera automatiquement adaptée en fonction. De cette façon, la dernière ligne visible sera toujours complète, ce qui n'est pas le cas en fixant les limites de la hauteur en nombre de points.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 MinLines="2" MaxLines="3"
 AcceptsReturn="True" AcceptsTab="True"
 TextWrapping="Wrap" />
```

Selon les comportements que vous aurez définis pour votre `TextBox`, l'utilisateur sera obligé de faire défiler le texte horizontalement et verticalement. Sans aide visuelle, il n'est pas toujours aisé pour l'utilisateur de voir le texte caché. Le mieux dans ce cas est d'ajouter des barres de défilement. À vous de choisir celle qu'il vous faut.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 MinLines="2" MaxLines="3" MaxWidth="300"
 AcceptsReturn="True" AcceptsTab="True"
 TextWrapping="NoWrap"
 VerticalScrollBarVisibility="Visible"
 HorizontalScrollBarVisibility="Visible" />
```

◄ **Figure 2-22 :** Défilement dans un TextBox

Les valeurs possibles pour les attributs `VerticalScrollBarVisibility` et `HorizontalScrollBarVisibility` sont `Auto`, `Hidden`, `Disabled` et `Visible`.

Renvoi *Pour plus d'informations sur l'utilisation des valeurs possibles, reportez-vous au paragraphe* ***Autoriser le défilement*** *page 77.*

Attention

Conflit entre hauteur et nombre de lignes

Si vous utilisez simultanément les attributs `MinLines`, `MaxLines` et les attributs `MinHeight`, `MaxHeight`, ce sont ces derniers qui auront la préséance. `MinLines` et `MaxLines` n'auront dès lors aucun effet.

Dans certaines circonstances, vous souhaiterez certainement afficher un `TextBox` mais sans autoriser l'utilisateur à en modifier le contenu. Pour ce faire, vous disposez de deux possibilités, soit utiliser l'attribut `IsReadOnly`, soit utiliser l'attribut `IsEnabled`. Chacune de ces méthodes offre ses avantages et ses inconvénients.

Voyons d'abord l'effet de `IsReadOnly`.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 Text="Ce texte est en lecture seule."
 IsReadOnly="True" />
```

◄ **Figure 2-23 :** Un TextBox en lecture seule

Visuellement, il n'y a aucune différence entre un `TextBox` limité à la lecture et un autre `TextBox`. Toutefois, il ne vous sera pas possible d'introduire du texte dans ce contrôle. En revanche, vous pourrez sélectionner du texte pour faire un copier en vue de le coller ailleurs.

Remarque

Valeur par défaut

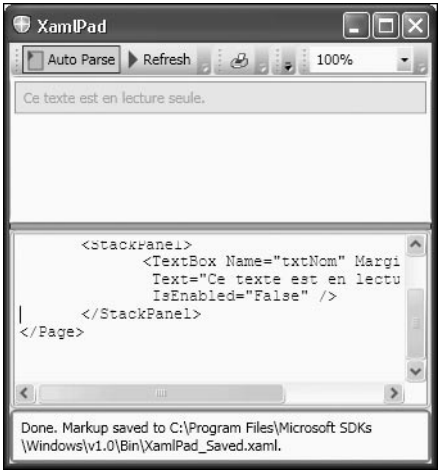
Il est difficile de concevoir une `TextBox` non modifiable sans valeur par défaut. Pour donner cette valeur, vous devez utiliser l'attribut `Text`.

En revanche, avec l'attribut `IsEnabled`, l'utilisateur visualise directement que le champ est inaccessible.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 Text="Ce texte est en lecture seule."
 IsEnabled="False" />
```

Cette technique a toutefois deux défauts. Premièrement, il n'est pas possible de sélectionner du texte en vue de le copier ailleurs. Deuxièmement, beaucoup d'utilisateurs se plaignent du manque de lisibilité des `TextBox` désactivés.

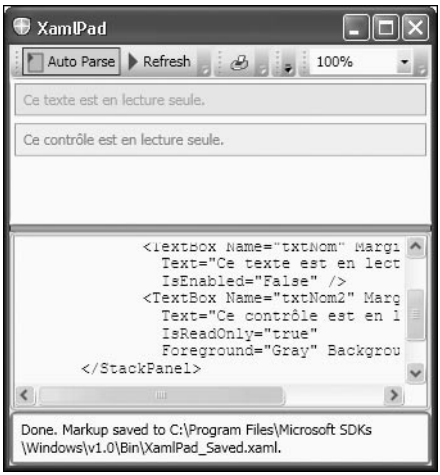
Vous pouvez modifier les attributs de couleur de votre `TextBox` pour qu'il affiche visuellement son état soit en copiant le style d'un `TextBox` désactivé, soit en utilisant votre propre charte graphique.



◄ **Figure 2-24 :** Un TextBox désactivé

Dans cet exemple, le style du champ en lecture seule est proche du style du champ désactivé tout en étant plus lisible.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
  Text="Ce texte est en lecture seule."
  IsEnabled="False" />
<TextBox Name="txtNom2" Margin="3,3,3,3"
  Text="Ce contrôle est en lecture seule."
  IsReadOnly="true"
  Foreground="Gray" Background="Beige" />
```



◄ **Figure 2-25 :** TextBox désactivé ou en lecture seule

Astuce

Introduire un mot de passe

Si vous désirez créer une zone de saisie pour introduire un mot de passe, utilisez la balise `PasswordBox` en lieu et place de `TextBox`. Il n'y a pas de différence majeure dans leur utilisation si ce n'est que les caractères affichés sont remplacés par des * et que la propriété `Text` est remplacée par la propriété `Password`.

2.3 Créer un bouton

Le bouton est également un contrôle indispensable, ne serait-ce que pour les célèbres boutons **OK** et **Annuler**.

Comme pour le `Label`, vous pouvez l'utiliser selon deux variantes légèrement différentes.

```
<Button Name="btnOk"
  Width="80" Height="30" >
   Ok
</Button>

<Button Name="btnCancel" Content="Annuler"
  Width="80" Height="30" />
```

◄ **Figure 2-26 :** Affichage de simples boutons

Remarque

Fixer la taille des boutons

Généralement, la taille des boutons présentés dans un même écran est fixe. Ne pas respecter cette règle produit le plus souvent un très mauvais effet visuel.

De manière générale, le bouton **OK** sera présenté comme bouton par défaut. De même manière, le bouton **Annuler** devrait être associé à la touche Echap. Pour qu'un bouton soit le bouton par défaut, il suffit d'utiliser l'attribut `IsDefault`. Pour associer un bouton à la touche Echap, il suffit d'utiliser l'attribut `IsCancel`.

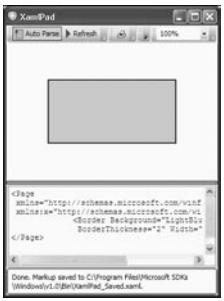
```
<Button Name="btnOk" IsDefault="True"
 Width="80" Height="30" >
   Ok
</Button>
<Button Name="btnCancel" Content="Annuler"
 Width="80" Height="30" IsCancel="True"/>
```

Nous devons encore pouvoir associer le bouton avec une action bien définie. Toutefois, cette opération faisant appel à du code .NET, nous reviendrons dessus dans le chapitre sur les événements (page 138).

2.4 Afficher un cadre

Comme nous l'avons vu précédemment, certains contrôles possèdent leurs propres attributs pour réaliser un cadre autour d'eux. Ce n'est toutefois pas le cas de tous. Le contrôle `Border` est là pour pallier ce problème. Ce contrôle est en définitive très couramment utilisé. Il permet de créer des cadres partout où vous en avez besoin.

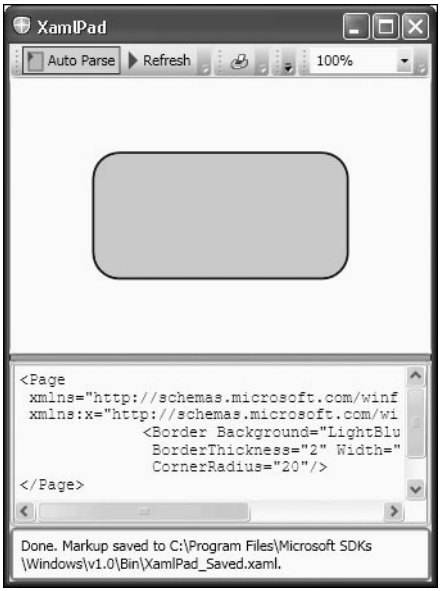
```
<Border Background="LightBlue" BorderBrush="DarkBlue"
 BorderThickness="2" Width="200" Height="100"/>
```



◄ **Figure 2-27 :** Créer un bord

Si vous souhaitez arrondir les coins, il suffit d'utiliser l'attribut CornerRadius.

```
<Border Background="LightBlue" BorderBrush="DarkBlue"
 BorderThickness="2" Width="200" Height="100"
 CornerRadius="20"/>
```

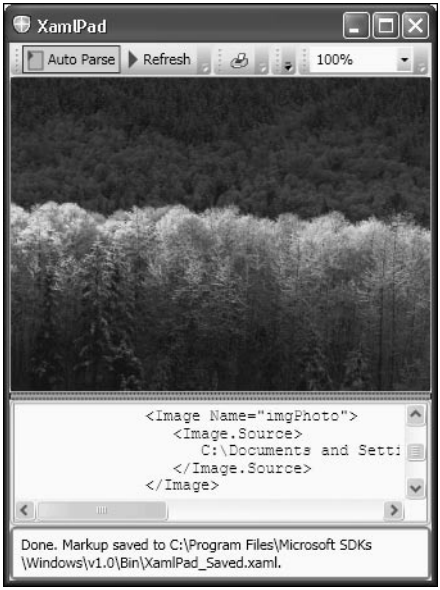


◄ **Figure 2-28** : Bord à coins arrondis

2.5 Afficher une image

Les images peuvent être utiles soit pour améliorer la présentation soit pour diffuser de l'information telle qu'une photo. Pour afficher une image avec XAML, nous utiliserons la balise Image.

```
<Image Name="imgPhoto">
   <Image.Source>
      C:\Documents and Settings\All Users\Documents\
      Mes images\Échantillons d'images\Hiver.jpg
   </Image.Source>
</Image>
```



◄ **Figure 2-29 :** Afficher une photo

Remarque

Erreur dans le code

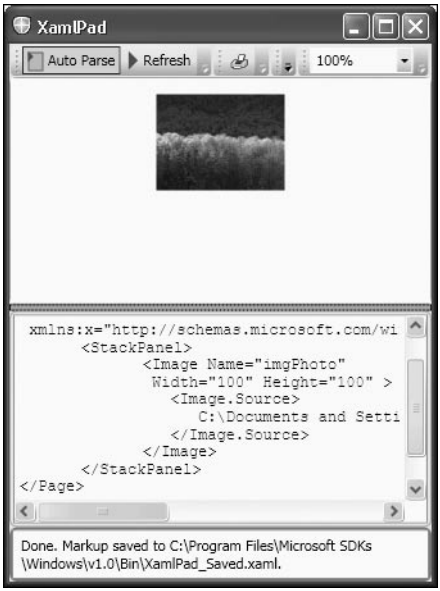
Le contenu de la balise `Image` est volontairement scindé en deux lignes pour une question de lisibilité de l'exemple. Toutefois, si vous désirez le reproduire, il sera nécessaire de regrouper le chemin du fichier sur une même ligne.

Le nœud fils `ImageSource` permet de définir le fichier qui sera affiché.

Les formats supportés sont **bmp**, **gif**, **ico**, **jpg**, **png** et **tiff**.

Vous pouvez également fixer les dimensions de l'image avec les différents attributs `MinWidth`, `MaxWidth`, `MinHeight`, `MaxHeight` ou encore, comme dans l'exemple ci-dessous, en faire une miniature en utilisant les attributs `Width` et `Height`.

```
<Image Name="imgPhoto"
 Width="100" Height="100" >
   <Image.Source>
      C:\Documents and Settings\All Users\Documents\
      Mes images\Échantillons d'images\Hiver.jpg
   </Image.Source>
</Image>
```



◄ **Figure 2-30 :** Afficher une miniature

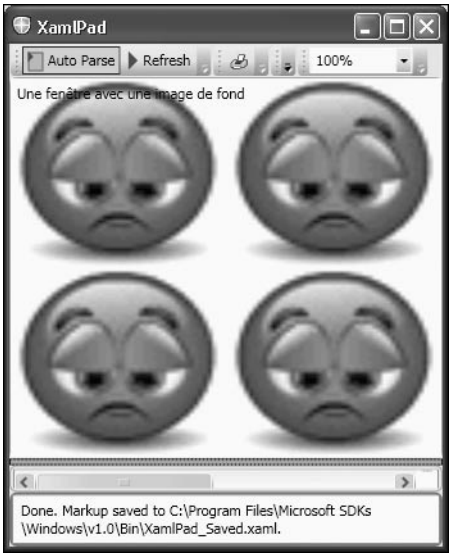
Il existe une seconde possibilité pour afficher des images mais il s'agit alors non pas directement d'un contrôle à placer dans votre page mais d'une image utilisée dans un contrôle ; par exemple insérer l'image comme fond d'un bouton, d'un cadre ou d'une page.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <Page.Background>
      <ImageBrush ImageSource="…\Emoticons\no.gif" />
   </Page.Background>
   <StackPanel>
      <Label>
         Une fenêtre avec une image de fond
      </Label>
   </StackPanel>
</Page>
```

◄ **Figure 2-31 :** Afficher une image comme fond de page

`ImageBrush` dispose d'attributs tout à fait particuliers pour gérer l'apparence de l'image. Il n'est pas question de taille puisque celle-ci dépend du conteneur. En revanche, il est toutefois possible de contrôler la taille relative en utilisant l'attribut `Viewport`. Si vous l'associez à `TileMode`, vous pouvez par exemple créer une mosaïque.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <Page.Background>
      <ImageBrush Viewport="0.5,0.5,0.5,0.5"
       TileMode="TILE"
       ImageSource="…\Emoticons\no.gif" />
   </Page.Background>
   <StackPanel>
      <Label>
         Une fenêtre avec une image de fond
      </Label>
   </StackPanel>
</Page>
```



◄ **Figure 2-32 :** Afficher une mosaïque comme fond de page

Il est aussi possible de paramétrer l'étirement de l'image et son alignement en utilisant les attributs `Stretch`, `AlignmentX`.et `AlignmentY`.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <Page.Background>
      <ImageBrush Stretch="None"
       AlignmentY="Bottom"
       AlignmentX="Right"
       ImageSource="…\Emoticons\no.gif" />
   </Page.Background>
   <StackPanel>
      <Label>
         Une fenêtre avec une image de fond
      </Label>
   </StackPanel>
</Page>
```

◄ **Figure 2-33 :** Afficher une image non étirée

2.6 Checklist

Voici les points essentiels que nous avons abordés dans ce chapitre :

- l'affichage des textes et leurs manipulations avec des `Label` et `TextBlock` ;
- l'affichage et la saisie des textes avec `TextBox` ;
- la saisie d'un mot de passe avec `PasswordBox` ;
- l'affichage des boutons avec `Button` ;
- l'affichage des images avec `Image` et `ImageBrush` ;
- la réalisation de cadres avec `Border` ;
- l'utilisation des attributs les plus communs et utilisables avec la majorité des contrôles ;
- l'utilisation des décorateurs avec `TextDecoration`.

Disposer les éléments à l'écran

Utiliser les coordonnées 56
Utiliser une grille 61
Mettre en page avec un WrapPanel 68
Utiliser un empilement 70
Utiliser le docking 72
Autoriser le défilement 77
Mélanger les techniques de mise en page 81
Créer une page composite 88
Checklist 90

Avec les contrôles vus au chapitre précédent, nous possédons assez d'éléments pour réaliser un écran complet et complexe. Ce qui nous manque maintenant, c'est une méthode pour organiser tout cela. XAML ne se contente pas d'une méthode mais nous en fournit plusieurs, chacune ayant ses avantages et ses inconvénients.

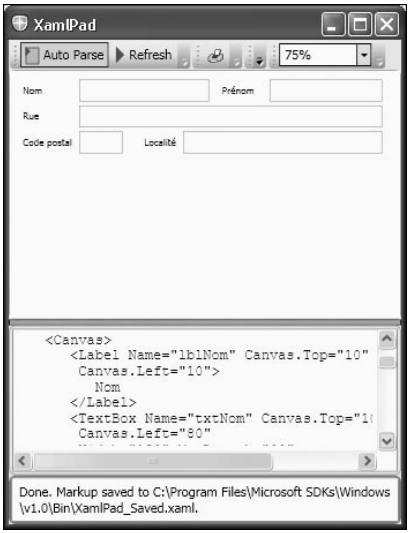
3.1 Utiliser les coordonnées

L'utilisation de `Canvas` est la méthode la plus proche de la technique utilisée avec les anciens API. Si vous avez déjà programmé sous Windows, vous savez certainement que les contrôles étaient positionnés relativement au coin supérieur gauche de la fenêtre. Pour positionner les contrôles dans un `Canvas`, c'est pareil.

Comme premier exemple, plaçons dans un écran des étiquettes et des boîtes de texte pour pouvoir introduire le nom, le prénom et l'adresse d'une personne.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Canvas>
      <Label Name="lblNom" Canvas.Top="10"
       Canvas.Left="10">
         Nom
      </Label>
      <TextBox Name="txtNom" Canvas.Top="10"
       Canvas.Left="80"
       Width="150" MaxLength="30"
       CharacterCasing="Upper" />
      <Label Name="lblPrenom" Canvas.Top="10"
       Canvas.Left="240">
         Prénom
      </Label>
      <TextBox Name="txtPrenom" Canvas.Top="10"
       Canvas.Left="300"
       Width="130" MaxLength="30"/>
      <Label Name="lblAdr" Canvas.Top="40"
       Canvas.Left="10">
         Rue
      </Label>
      <TextBox Name="txtAdr" Canvas.Top="40"
       Canvas.Left="80"
       Width="350"  MaxLength="80"/>
      <Label Name="lblCP" Canvas.Top="70"
       Canvas.Left="10">
         Code postal
      </Label>
```

```
      <TextBox Name="txtCP" Canvas.Top="70"
       Canvas.Left="80"
       Width="50" MaxLength="5"/>
      <Label Name="lblLocalite" Canvas.Top="70"
       Canvas.Left="150">
         Localité
      </Label>
      <TextBox Name="txtLocalite" Canvas.Top="70"
       Canvas.Left="200"
       Width="230"  MaxLength="50"/>
   </Canvas>
</Page>
```



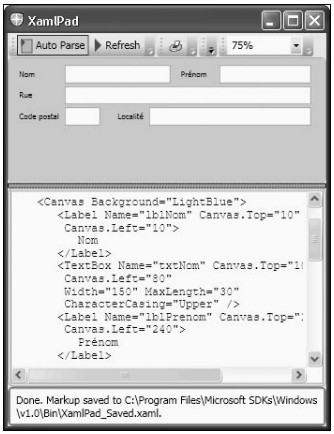
◄ **Figure 3-1 :** Présentation avec un Canvas

Comme vous pouvez le constater, le placement à l'écran se fait au moyen des attributs `Canvas.Top` et `Canvas.Left`. Il s'agit de propriétés attachées. Une propriété attachée est en fait une propriété du parent, ici du `Canvas`, mais pour laquelle chaque enfant peut assigner une valeur différente. C'est pourquoi, bien que propriété de `Canvas`, l'attribut `Canvas.Top` est par exemple un attribut des éléments contenus dans le `Canvas` et non du `Canvas` lui-même.

La balise `Canvas` dispose elle-même de quelques attributs.

L'attribut `Background` vous permet de changer la couleur du fond si la couleur standard ne vous satisfait pas.

```
<Canvas Background="LightCoral">
```



◄ **Figure 3-2 :** La couleur du fond d'un Canvas

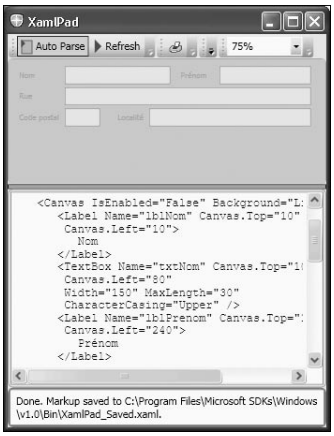
Un autre attribut intéressant est l'attribut `IsEnabled`. En effet, si vous lui assignez la valeur `False`, l'ensemble des contrôles contenus dans le `Canvas` seront eux aussi désactivés.

Attention

Désactivé et non lecture seule

N'oubliez pas toutefois les différences que nous avons vues précédemment entre désactiver et lecture seule pour un contrôle tel que la boîte de texte.

```
<Canvas IsEnabled="False" Background="LightCoral">
```



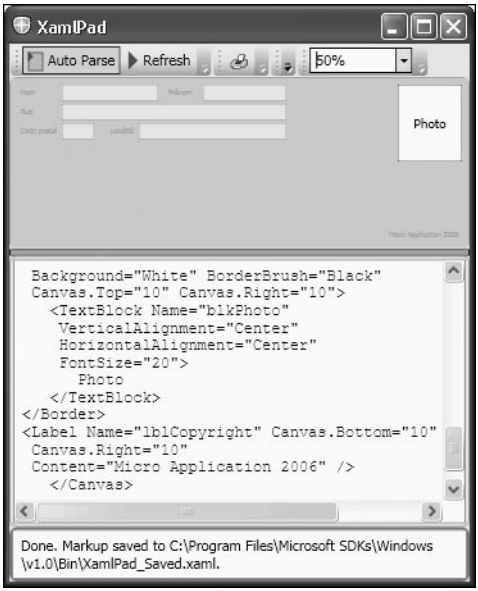
◄ **Figure 3-3 :** Un Canvas désactivé

Les attributs `Canvas.Top` et `Canvas.Left` ne sont pas les seuls utilisables pour positionner un contrôle. Vous pouvez également utiliser `Canvas.Right` et `Canvas.Bottom`, ce qui permet de placer un contrôle relativement à n'importe quel coin du `Canvas`.

Ajoutons le code suivant devant la balise de fin du nœud `Canvas`.

```
<Border Width="100" Height="120" BorderThickness="1"
 Background="White" BorderBrush="Black"
 Canvas.Top="10" Canvas.Right="10">
   <TextBlock Name="blkPhoto"
    VerticalAlignment="Center"
    HorizontalAlignment="Center"
    FontSize="20">
      Photo
   </TextBlock>
</Border>
<Label Name="lblCopyright" Canvas.Bottom="10"
 Canvas.Right="10"
 Content="Micro Application 2006" />
```

◄ **Figure 3-4 :** Un Canvas avec différents points de référence



◄ **Figure 3-5 :** Le même Canvas agrandi

Comme vous pouvez le constater, le fait de positionner un contrôle non pas à partir du coin supérieur gauche mais d'un autre coin ne fait pas que changer le système de coordonnées. Lors du redimensionnement de l'écran, les contrôles suivent le coin à partir duquel ils sont positionnés. Cette faculté pourra être utilisée dans de nombreuses circonstances pour obtenir un écran beaucoup plus flexible.

Astuce

Fixer une taille minimale à son Canvas

Si vous ne fixez pas une taille minimale à votre `Canvas`, les éléments contenus vont inévitablement se chevaucher si la fenêtre est réduite au-delà de la taille critique.

En remplaçant la balise `Canvas` dans l'exemple par celle ci-dessous, nous empêcherons ce comportement chaotique.

```
<Canvas Background="LightBlue"
 MinWidth="550"
 MinHeight="200">
```

3.2 Utiliser une grille

Cette méthode est généralement la méthode la plus recommandée car elle offre un très haut niveau d'adaptabilité du contenu de l'écran à sa taille et donc également aux changements de résolution. Elle est toutefois plus complexe à mettre en œuvre.

L'idée est de placer un contrôle par cellule de la grille. Il y a donc lieu de bien définir la grille pour obtenir le résultat voulu. Essayons de reproduire l'exemple précédent mais avec la balise `Grid` au lieu de `Canvas`.

L'objectif de l'utilisation d'une grille étant une plus grande adaptabilité, nous ne fixerons pas la taille des contrôles.

Si nous étudions l'écran tel qu'il est désiré, nous pouvons en déduire qu'il nécessite 4 lignes et 5 colonnes.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Grid>
```

La première chose à faire lorsque l'on utilise une grille, c'est de définir les lignes et les colonnes.

```
      <Grid.ColumnDefinitions>
         <ColumnDefinition />
         <ColumnDefinition />
         <ColumnDefinition />
         <ColumnDefinition />
         <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition />
         <RowDefinition />
         <RowDefinition />
         <RowDefinition />
      </Grid.RowDefinitions>
```

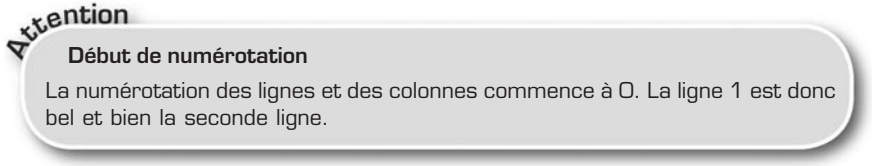
Nous pouvons ensuite utiliser les contrôles tout en définissant leurs positions dans la grille en utilisant `Grid.Row` et `Grid.Column`.

```
      <Label Name="lblNom" Grid.Row="0" Grid.Column="0">
         Nom
      </Label>
      <TextBox Name="txtNom" Grid.Row="0" Grid.Column="1"
       MaxLength="30" CharacterCasing="Upper" />
      <Label Name="lblPrenom" Grid.Row="0" Grid.Column="2">
```

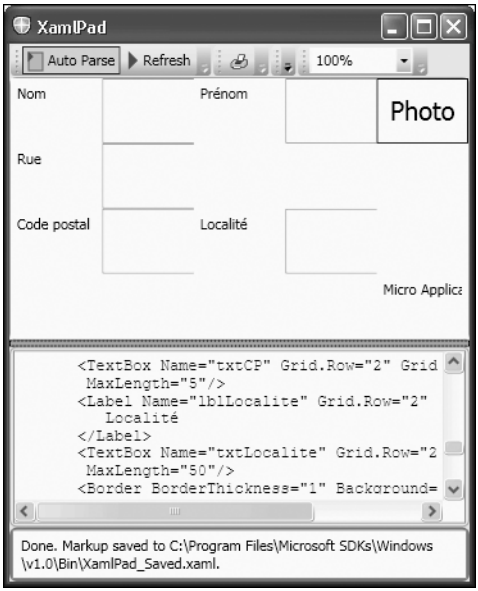
```
        Prénom
      </Label>
      <TextBox Name="txtPrenom" Grid.Row="0" Grid.Column="3"
       MaxLength="30"/>
      <Label Name="lblAdr" Grid.Row="1" Grid.Column="0">
        Rue
      </Label>
      <TextBox Name="txtAdr" Grid.Row="1" Grid.Column="1"
       MaxLength="80"/>
      <Label Name="lblCP" Grid.Row="2" Grid.Column="0">
        Code postal
      </Label>
      <TextBox Name="txtCP" Grid.Row="2" Grid.Column="1"
       MaxLength="5"/>
      <Label Name="lblLocalite" Grid.Row="2" Grid.Column="2" >
        Localité
      </Label>
      <TextBox Name="txtLocalite" Grid.Row="2" Grid.Column="3"
       MaxLength="50"/>
      <Border BorderThickness="1" Background="White"
       Grid.Row="0" Grid.Column="4" BorderBrush="Black">
        <TextBlock Name="blkPhoto" VerticalAlignment="Center"
         HorizontalAlignment="Center" FontSize="20">
          Photo
        </TextBlock>
      </Border>
      <Label Name="lblCopyright"
       Content="Micro Application 2006"
        Grid.Row="3" Grid.Column="4"/>
   </Grid>
</Page>
```

Comme vous pouvez le constater, la définition de la grille se fait au début en utilisant les collections `RowDefinitions` et `ColumnDefinitions`. Ces collections contiennent autant d'éléments que de lignes ou de colonnes désirées.

À la place des attributs `Canvas.Top` et autre `Canvas.Left`, ce sont les attributs `Grid.Column` et `Grid.Row`.



Malheureusement, comme vous pouvez le voir ci-dessous, le résultat obtenu n'est pas vraiment à la hauteur des espérances.



◄ **Figure 3-6 :** Affichage avec une grille

Commençons par le plus simple, le fond d'écran n'est pas coloré. Pour cela, pas de soucis, il suffit d'utiliser le même attribut que pour `Canvas`.

```
<Grid Background="LightBlue">
```

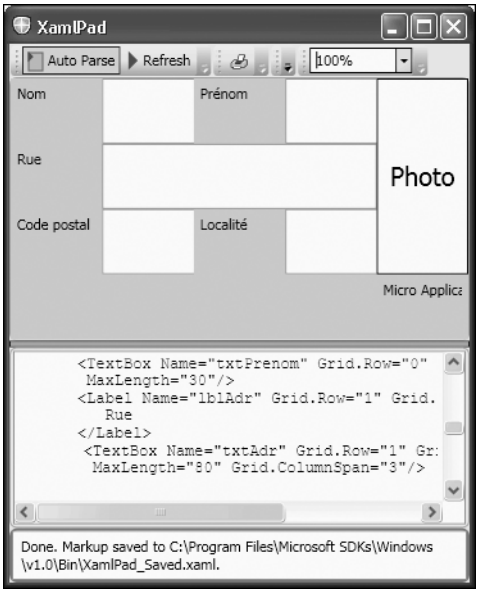
Autre problème évident, la zone de saisie de l'adresse est limitée à la deuxième colonne. Pour résoudre ce problème, il est possible d'étendre le contrôle sur plusieurs colonnes en utilisant l'attribut `Grid.ColumnSpan`.

```
<TextBox Name="txtAdr" Grid.Row="1" Grid.Column="1"
 MaxLength="80" Grid.ColumnSpan="3"/>
```

Il en va de même pour la zone photo qui est limitée à une ligne. La solution est quasiment identique excepté que l'attribut est `Grid.RowSpan`.

```
<Border BorderThickness="1" Background="White"
 Grid.Row="0" Grid.Column="4" Grid.RowSpan="3"
 BorderBrush="Black">
   <TextBlock Name="blkPhoto" VerticalAlignment="Center"
    HorizontalAlignment="Center" FontSize="20">
      Photo
   </TextBlock>
</Border>
```

Avec l'ajout de ces nouveaux attributs, l'écran est enfin un peu plus ressemblant.



◄ **Figure 3-7** : Le même exemple un peu plus complet

Quelques aménagements sont encore nécessaires pour parfaire le travail.

```
<Label Name="lblCopyright"
 Content="Micro Application 2006"
 Grid.Row="3" Grid.Column="3"
 Grid.ColumnSpan="2"
 VerticalAlignment="Bottom"
 HorizontalAlignment="Right"/>
```

Le code ainsi modifié va permettre de placer le copyright en bas à droite de l'écran, et ce quelle que soit sa dimension comme c'était le cas avec le canevas.

Placez également l'attribut ci-dessous dans chaque balise `TextBox` et dans la balise `Border`.

```
Margin="2,2,2,2"
```

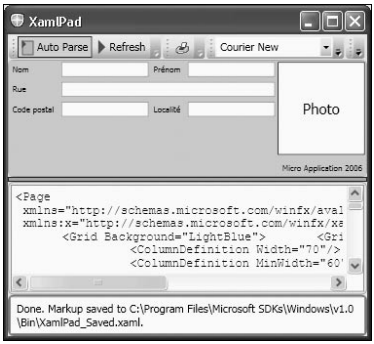
Le gros problème restant est l'élargissement inconsidéré des boîtes de texte, qui dénature fortement l'effet visuel de l'écran. Pour y remédier, il suffit d'assigner les attributs `MaxHeight` et `VerticalAlignment` à chaque `TextBox`.

Par exemple en les fixant aux valeurs suivantes :

```
MaxHeight="20" VerticalAlignment="Top"
```

Pour contrôler le comportement de la fenêtre en cas de redimensionnement, vous pouvez ajuster les dimensions des lignes et des colonnes. N'hésitez pas à utiliser les dimensions minimales et maximales.

```
<Grid.ColumnDefinitions>
   <ColumnDefinition Width="70"/>
   <ColumnDefinition MinWidth="60"/>
   <ColumnDefinition Width="50"/>
   <ColumnDefinition MinWidth="60"/>
   <ColumnDefinition MinWidth="120" MaxWidth="200"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
   <RowDefinition MinHeight="22"/>
   <RowDefinition MinHeight="22"/>
   <RowDefinition MinHeight="76"/>
   <RowDefinition MinHeight="20"/>
</Grid.RowDefinitions>
```



◄ **Figure 3-8 :** L'exemple complet

Redimensionnez la fenêtre et voyez ce qui se passe.

Bien sûr, comme pour `Canvas`, vous pouvez utiliser l'attribut `IsEnabled` pour désactiver l'ensemble de la grille.

Astuce

Visualiser la grille

Parfois, pour faciliter la mise en page, il est intéressant de voir les limites des cellules. Pour cela, utilisez l'attribut `ShowGridLines`.

En complément de ces possibilités, XAML offre la possibilité de créer des bords mobiles pour permettre à l'utilisateur d'ajuster la taille que ce soit en largeur ou en hauteur. Cette possibilité doit être mise en œuvre en utilisant un `GridSplitter`. Cette balise va vous permettre de définir dans la grille sur quelle cellule ou groupe de cellules vous désirez placer la bordure mobile et dans quel sens. Afin de vous montrer cette fonctionnalité, nous allons construire un tableau de 3 lignes et 4 colonnes. La hauteur de la première ligne pourra être adaptée ainsi que la largeur de la première colonne. La première partie du code est classique. Notez toutefois que 4 lignes et non 3 sont définies. Nous reviendrons sur le pourquoi à la fin de l'exercice. Ensuite, les 2 `GridSplitter` sont définis. Le premier dans le sens vertical, *via* l'attribut `ResizeDirection= "Columns"`, l'autre dans le sens horizontal avec `ResizeDirection= "Rows"`. La position est donnée respectivement avec `Grid.Column` et `Grid.Row`. `ColSpan` et `RowSpan` permettent d'étendre la visibilité du bord sur l'ensemble des cellules.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition />
         <ColumnDefinition/>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition/>
         <RowDefinition Height="5"/>
         <RowDefinition/>
         <RowDefinition/>
      </Grid.RowDefinitions>
      <GridSplitter Grid.Column="0" Grid.RowSpan="4"
       ResizeDirection="Columns" ShowsPreview="False"/>
      <GridSplitter Grid.Row="1" Grid.ColumnSpan="4"
       ResizeDirection="Rows" HorizontalAlignment="Stretch"
       ShowsPreview="True"/>
   </Grid>
</Page>
```

◄ **Figure 3-9 :** Les bords mobiles

L'attribut `ShowsPreview` modifie le comportement de telle sorte que, si vous souhaitez la prévisualisation, le bord ne bouge que quand vous lâchez le bouton de la souris. Dans l'intervalle, c'est une ligne repère qui prévisualise la future position du bord.

◄ **Figure 3-10 :** Les bords mobiles avec prévisualisation

Astuce

4 lignes au lieu de 3

Maintenant, comme promis, pourquoi 4 et non 3 lignes. Comme vous avez pu le constater, la hauteur de la ligne a été fixée à 5. En fait, cette astuce est utilisée pour pallier un comportement différent entre le bord horizontal et vertical. Cette différence aura probablement disparu dans la version définitive et

Astuce

cette ligne supplémentaire n'aura plus lieu d'être mais, dans le doute, il vaut mieux savoir comment contourner le problème. Donc, si pour un bord vertical celui-ci se place correctement à droite de la colonne, pour une ligne horizontale il n'apparaît pas. En utilisant `HorizontalAlignment= « Stretch »`, le bord devient visible mais occupe toute la hauteur. L'idée est donc d'utiliser une ligne supplémentaire pour contenir le bord ainsi créé.

3.3 Mettre en page avec un WrapPanel

La mise en page avec un `WrapPanel` n'est pas du tout adaptée à l'exemple précédent. En effet, avec un `WrapPanel` les contrôles sont placés à côté les uns des autres et sont renvoyés automatiquement à la ligne lorsque la fin de celle-ci est atteinte.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <WrapPanel>
      <TextBlock Width="200" TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
         <Bold>WrapPanel</Bold>
         La mise en page avec un WrapPanel n'est pas du tout
         adaptée à l'exemple précédent. En effet avec un
         WrapPanel les contrôles sont placés à coté les un
         des autres et sont renvoyés automatiquement à la
         ligne lorsque la fin de celle-ci est atteinte.
      </TextBlock>
```

Les différents contrôles doivent être indiqués dans l'ordre dans lequel vous désirez les voir apparaître. Ils sont affichés de gauche à droite et de haut en bas.

```
      <TextBlock Width="200" TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
       <Bold>StackPanel</Bold>
       L'utilisation d'un StackPanel offre encore des
       possibilités plus restreinte que le WrapPanel.
       Toutefois, contrairement au WrapPanel, il offre
       une structure moins mobile et donc beaucoup plus
       controlable. Dans un StackPanel, chaque contrôle occupe
       une ligne.
      </TextBlock>
      <TextBlock Width="200" TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
```

```
        <Bold>Grid</Bold>: Cette méthode est généralement
        la méthode la plus recommandée car elle offre un très
        haut niveau d'adaptabilité du contenu de l'écran à sa
        taille et donc également aux changements de
        résolution. Elle est toutefois plus complexe à mettre
        en œuvre.
      </TextBlock>
   </WrapPanel>
</Page>
```

◄ **Figure 3-11 :** Utilisation d'un WrapPanel

Si vous redimensionnez la fenêtre, le `WrapPanel` se charge de replacer les contrôles.

◄ **Figure 3-12 :** Le même exemple redimensionné

3.4 Utiliser un empilement

L'utilisation d'un `StackPanel` offre encore des possibilités plus restreintes que le `WrapPanel`. Toutefois, contrairement au `WrapPanel`, il offre une structure moins mobile et donc beaucoup plus facile à contrôler. Dans un `StackPanel`, chaque contrôle occupe une ligne.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <StackPanel>
      <TextBlock TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
         <Bold>WrapPanel</Bold>
         La mise en page avec un WrapPanel n'est pas du
         tout adaptée à l'exemple précédent. En effet avec
         un WrapPanel les contrôles sont placés à coté les
         un des autres et sont renvoyés automatiquement à
         la ligne lorsque la fin de celle-ci est atteinte.
      </TextBlock>
```

Les différents contrôles doivent être indiqués dans l'ordre dans lequel vous désirez les voir apparaître. Ils sont affichés de haut en bas.

```
      <TextBlock TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
       <Bold>StackPanel</Bold>
       L'utilisation d'un StackPanel offre encore des
       possibilités plus restreinte que le WrapPanel.
       Toutefois, contrairement au WrapPanel, il offre
       une structure moins mobile et donc beaucoup plus
       controlable. Dans un StackPanel, chaque contrôle
       occupe une ligne.
      </TextBlock>
      <TextBlock TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" TextAlignment="Justify">
         <Bold>Grid</Bold>: Cette méthode est généralement
         la méthode la plus recommandée car elle offre un très
         haut niveau d'adaptabilité du contenu de l'écran à sa
         taille et donc également aux changements de
         résolution. Elle est toutefois plus complexe à mettre
         en œuvre.
      </TextBlock>
   </StackPanel>
</Page>
```

◄ **Figure 3-13 :** Utilisation d'un StackPanel

◄ **Figure 3-14 :** Le même code

Remarque

Longueur de ligne

Contrairement à l'exemple tel qu'il est dans le chapitre sur le `WrapPanel`, où nous avions défini la largeur avec la propriété `Width`, comme avec `StackPanel` les éléments sont automatiquement placés l'un sous l'autre il n'est pas nécessaire de fixer la largeur.

3.5 Utiliser le docking

Avec un DockPanel vous aurez la possibilité de coller vos contrôles sur les différents bords. Comme rien ne vaut un exemple, regardons le code ci-dessous, qui est dérivé des exemples précédents.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <DockPanel>
```

Le contrôle suivant sera collé sur le bord supérieur.

```
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Top" Margin="2,2,2,2" >
         <TextBlock TextWrapping="WrapWithOverflow"
          Margin="5,5,5,5" TextAlignment="Justify">
            <Bold>WrapPanel</Bold>
            La mise en page avec un WrapPanel n'est pas
            du tout adaptée à l'exemple précédent.
            En effet, avec un WrapPanel les contrôles sont
            placés à côté les uns des autres et sont renvoyés
            automatiquement à la ligne lorsque la fin
            de celle-ci est atteinte.
         </TextBlock>
      </Border>
```

Ce contrôle-ci sera en revanche collé sur le bord droit.

```
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Right" Margin="2,2,2,2" Width="200">
         <TextBlock TextWrapping="WrapWithOverflow"
          Margin="5,5,5,5" TextAlignment="Justify">
            <Bold>StackPanel</Bold>
            L'utilisation d'un StackPanel offre encore des
            possibilités plus restreintes que le WrapPanel.
            Toutefois, contrairement au WrapPanel, il offre
            une structure moins mobile et donc beaucoup plus
            contrôlable. Dans un StackPanel, chaque contrôle
            occupe une ligne.
         </TextBlock>
      </Border>
```

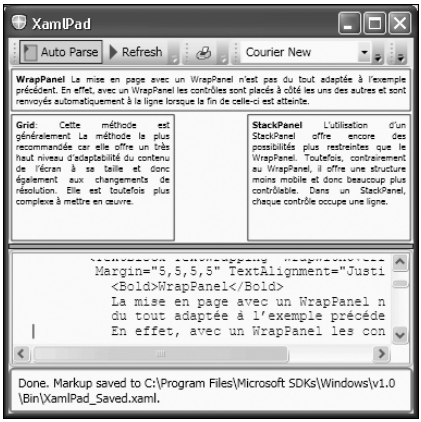
Ce dernier contrôle sera quant à lui collé sur le bord gauche.

```
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Left" Width="200" Margin="2,2,2,2"
       HorizontalAlignment="Left">
```

```
        <TextBlock TextWrapping="WrapWithOverflow"
         Margin="5,5,5,5" TextAlignment="Justify">
           <Bold>Grid</Bold>: Cette méthode est généralement
           La méthode la plus recommandée car elle offre un
           très haut niveau d'adaptabilité du contenu de
           l'écran à sa taille et donc également aux
           changements de résolution. Elle est toutefois plus
           complexe à mettre en œuvre.
        </TextBlock>
      </Border>
   </DockPanel>
</Page>
```

Tout d'abord, des balises `Border` ont été ajoutées afin de mieux visualiser le résultat. Elles n'ont aucun but particulier si ce n'est une meilleure compréhension du résultat. Du fait de leur présence, la taille des différents blocs est fixée dans les balises `Border` et non plus dans les balises `TextBlock`. Les marges sont également présentes pour bien différencier les divers éléments.

Voyons maintenant le résultat.

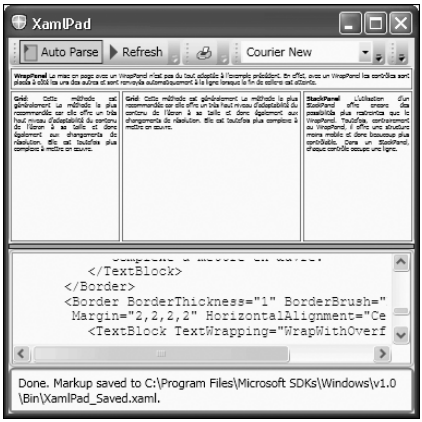


◄ **Figure 3-15 :** Exemple d'utilisation d'un DockPanel

Si nous ajoutons un cadre, celui-ci prendra automatiquement toute la place restant disponible.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <DockPanel>
…
      <Border BorderThickness="1" BorderBrush="Black"
       Margin="2,2,2,2" HorizontalAlignment="Center">
```

```
        <TextBlock TextWrapping="WrapWithOverflow"
         Margin="5,5,5,5" TextAlignment="Justify">
           <Bold>Grid</Bold>: Cette méthode est généralement
           La méthode la plus recommandée car elle offre un
           très haut niveau d'adaptabilité du contenu de
           l'écran à sa taille et donc également aux
           changements de résolution. Elle est toutefois plus
           complexe à mettre en œuvre.
        </TextBlock>
      </Border>
   </DockPanel>
</Page>
```

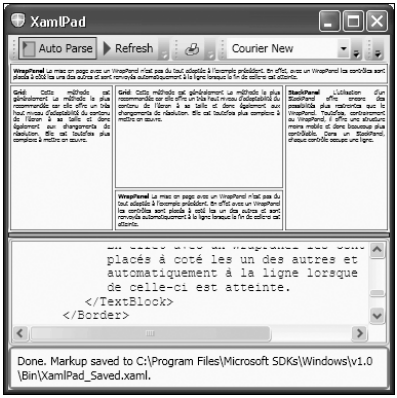


◄ **Figure 3-16 :** Exemple d'utilisation d'un DockPanel

L'ordre des contrôles a une importance capitale quant au résultat obtenu. Si nous ajoutons un cinquième cadre et que nous fixions l'attribut `DockPanel.Dock` à `Bottom`, le cadre n'occupera pas l'entièreté du bas d'écran comme c'est le cas pour le cadre du haut.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <DockPanel>
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Top" Margin="2,2,2,2" >
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
      <Border BorderThickness="1" BorderBrush="Black"
```

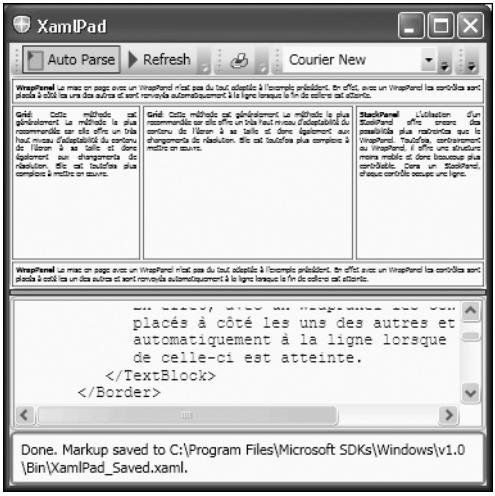
```
        DockPanel.Dock="Right" Margin="2,2,2,2" Width="200">
          <TextBlock TextWrapping="WrapWithOverflow"
…
          </TextBlock>
        </Border>
        <Border BorderThickness="1" BorderBrush="Black"
         DockPanel.Dock="Left" Width="200" Margin="2,2,2,2"
         HorizontalAlignment="Left">
          <TextBlock TextWrapping="WrapWithOverflow"
…
          </TextBlock>
        </Border>
        <Border BorderThickness="1" BorderBrush="Black"
         DockPanel.Dock="Bottom" Margin="2,2,2,2" >
          <TextBlock TextWrapping="WrapWithOverflow"
           Margin="5,5,5,5" TextAlignment="Justify">
            <Bold>WrapPanel</Bold>
            La mise en page avec un WrapPanel n'est pas
            du tout adaptée à l'exemple précédent.
            En effet avec un WrapPanel les contrôles sont
            placés à coté les un des autres et sont renvoyés
            automatiquement à la ligne lorsque la fin
            de celle-ci est atteinte.
          </TextBlock>
        </Border>
        <Border BorderThickness="1" BorderBrush="Black"
         Margin="2,2,2,2" HorizontalAlignment="Center">
          <TextBlock TextWrapping="WrapWithOverflow"
…
          </TextBlock>
        </Border>
      </DockPanel>
</Page>
```



◄ **Figure 3-17 :** Exemple d'utilisation d'un DockPanel

En revanche, si dans le code nous déplaçons ce dernier cadre avant les cadres de gauche et de droite, elle occupe alors toute la largeur.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <DockPanel>
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Top" Margin="2,2,2,2" >
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Bottom" Margin="2,2,2,2" >
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Right" Margin="2,2,2,2" Width="200">
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
      <Border BorderThickness="1" BorderBrush="Black"
       DockPanel.Dock="Left" Width="200" Margin="2,2,2,2"
       HorizontalAlignment="Left">
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
      <Border BorderThickness="1" BorderBrush="Black"
       Margin="2,2,2,2" HorizontalAlignment="Center">
         <TextBlock TextWrapping="WrapWithOverflow"
…
         </TextBlock>
      </Border>
   </DockPanel>
</Page>
```



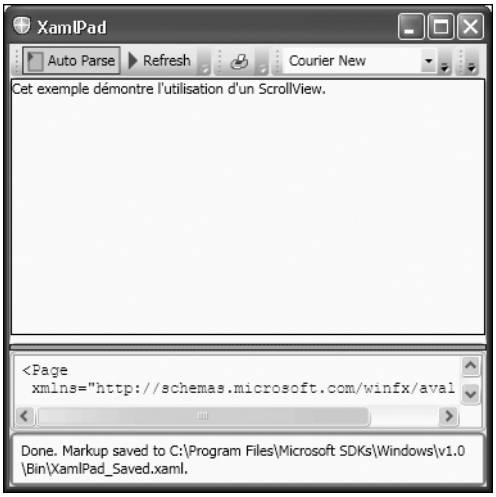
◄ **Figure 3-18 :** Exemple d'utilisation d'un DockPanel

Suivant cette même technique, vous pouvez ordonner les contrôles pour obtenir la segmentation qui vous convient.

3.6 Autoriser le défilement

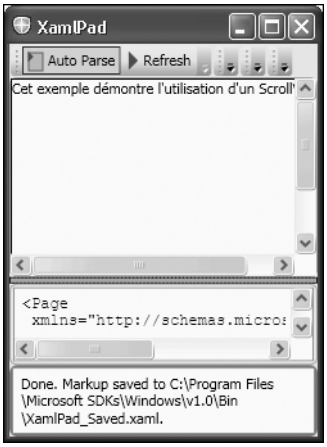
Comme vous l'aurez probablement déjà remarqué, lorsque la fenêtre ne peut plus contenir l'ensemble des éléments, les barres de défilement n'apparaissent pas sur les bords. Le `ScrollViewer` est là pour pallier ce problème.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <ScrollViewer
    ScrollViewer.HorizontalScrollBarVisibility="Auto"
    ScrollViewer.VerticalScrollBarVisibility="Auto">
      <StackPanel>
         <Border MinWidth="200" MinHeight="200"
          BorderBrush="Black" BorderThickness="1">
            <TextBlock TextWrapping="Wrap"
             Margin="0,0,0,20">
               Cet exemple démontre l'utilisation
               d'un ScrollView.
            </TextBlock>
         </Border>
      </StackPanel>
   </ScrollViewer>
</Page>
```



◄ **Figure 3-19 :** Exemple d'utilisation d'un ScrollViewer

Si nous réduisons la fenêtre, les barres de défilement apparaissent.



◄ **Figure 3-20 :** Barres de défilement en action

La balise `ScrollViewer` gère à la fois la barre horizontale et verticale. Dans l'exemple, les attributs `ScrollViewer.HorizontalScrollBarVisibility` et `ScrollViewer.VerticalScrollBarVisibility` sont initialisés à `Auto`. Cela correspond à l'usage le plus fréquent pour les barres de défilement. En mode automatique, les barres ne sont visibles que lorsqu'elles deviennent obligatoires.

Les autres valeurs possibles sont `Disabled`, qui a pour effet de ne pas avoir de barre de défilement dans la direction correspondante ; `Hidden`, qui n'affiche pas

la barre de défilement, mais le contenu se comporte comme si celle-ci était présente, c'est-à-dire qu'il n'y aura pas de passage à la ligne automatique ; la dernière valeur possible est `Visible`, qui fonctionne comme `Auto` à la différence près que la barre de défilement est visible même si elle n'est pas utile. Dans ce dernier cas, elle sera grisée.

```
<ScrollViewer
    ScrollViewer.HorizontalScrollBarVisibility="Visible"
    ScrollViewer.VerticalScrollBarVisibility="Visible">
```

◄ **Figure 3-21 :** Barres de défilement grisées

Remarque

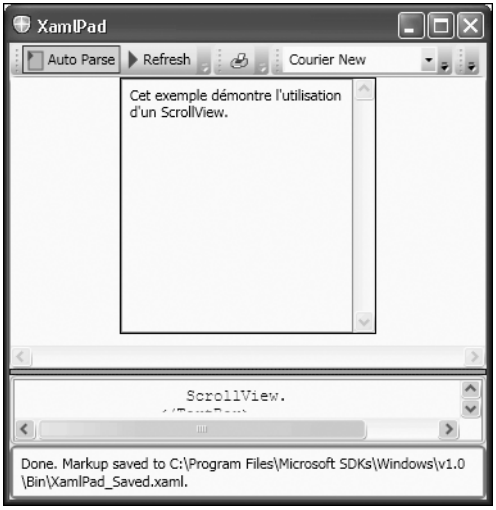
Valeur par défaut

Si vous ne spécifiez que la balise `ScrollViewer`, la barre horizontale sera cachée et inactive (`Hidden`) alors que la barre verticale sera visible (`Visible`).

L'utilisation d'un `ScrollViewer` ne se limite pas au bord de la fenêtre. Vous pouvez parfaitement en intégrer d'autres dans l'interface.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <ScrollViewer
    ScrollViewer.HorizontalScrollBarVisibility="Visible"
```

```
    ScrollViewer.VerticalScrollBarVisibility="Hidden">
      <StackPanel>
        <Border Width="200" Height="200"
          BorderBrush="Black" BorderThickness="1">
          <ScrollViewer
           ScrollViewer.HorizontalScrollBarVisibility="Disabled"
           ScrollViewer.VerticalScrollBarVisibility="Visible">
            <TextBox TextWrapping="Wrap"
              AcceptsReturn="True">
               Cet exemple démontre l'utilisation d'un
               ScrollView.
            </TextBox>
          </ScrollViewer>
        </Border>
      </StackPanel>
    </ScrollViewer>
</Page>
```



◄ **Figure 3-22 :** ScrollViewer sur un contrôle

Tapez du texte dans la boîte de texte et vous verrez apparaître la barre de défilement.

3.7 Mélanger les techniques de mise en page

Toutes les méthodes que nous venons de voir permettent chacune de résoudre certaines situations mais, bien souvent, vous vous trouvez en face d'un problème où vous devriez, pour partie, utiliser telle méthode et pour partie telle autre méthode. En XAML, vous pouvez utiliser autant de méthodes de mise en page que vous voulez, et cela dans le même écran. Bien sûr, pour y arriver, il faut respecter quelques règles.

Le principe est simple. Il s'agit ni plus ni moins que le principe de la poupée russe. Si une page ne peut contenir qu'un seul objet, disons un `Grid`, celui-ci peut contenir des contrôles mais également un autre conteneur comme un `Canvas`. Le `Canvas` peut à son tour contenir un `Grid` ou un `DockPanel`, par exemple.

Il n'y a quasiment plus de limite pour réaliser la mise en page de vos rêves. Mais, attention, plus complexe sera votre écran, plus complexe sera votre structure et plus la réalisation sera ardue et longue !

De fait, il existe non pas une méthode pour réaliser la mise en page mais une infinité de méthodes avec chacune leurs avantages et leurs inconvénients.

L'exemple qui suit est à titre purement éducatif et aurait vraisemblablement pu être résolu différemment. Toutefois, son objectif est purement didactique et, de ce point de vue, il remplit pleinement son rôle.

Nous allons au travers de cet exemple non seulement voir comment intégrer différents conteneurs mais également en profiter pour à nouveau voir certaines fonctionnalités des contrôles courants.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
```

Nous commençons par définir une barre de défilement horizontale et une verticale.

```
   <ScrollViewer
    ScrollViewer.HorizontalScrollBarVisibility="Visible"
    ScrollViewer.VerticalScrollBarVisibility="Visible">
```

Le premier conteneur défini est un `DockPanel`. Il va nous permettre de découper la fenêtre en quatre zones. Une pour l'en-tête, une pour le bas de page et deux pour le corps de la fenêtre.

```
      <DockPanel>
```

Tout d'abord l'en-tête, qui contient un Canvas.

```
<Border BorderThickness="2" DockPanel.Dock="Top"
 BorderBrush="LightGreen">
   <Canvas Background="DarkSeaGreen"
    MinHeight="40" >
      <Label FontSize="24" FontFamily="Engravers MT"
       Content="Ma bibliothèque" />
   </Canvas>
</Border>
```

Ensuite le bas d'écran, qui de façon qu'il occupe toute la largeur de la fenêtre doit être défini à ce moment. Il contient également un Canvas dans lequel le contrôle est positionné depuis le coin inférieur droit.

```
<Canvas DockPanel.Dock="Bottom"
 Background="Green" MinHeight="30">
   <Label Name="lblCopyright"
    Content="Le concepteur 2006"
    Canvas.Bottom="3" Canvas.Right="10"
    VerticalAlignment="Bottom"
    HorizontalAlignment="Right"/>
</Canvas>
```

La partie droite du corps contient un WrapPanel. Cette partie sera fixe lors d'un redimensionnement horizontal.

```
<Border DockPanel.Dock="Right"
 MinWidth="200" MaxWidth="400"
 BorderThickness="1" BorderBrush="Black"
 Background="LightGreen">
   <WrapPanel>
      <Image Width="190" Margin="2,2,2,2">
         <Image.Source>
            C:\Documents and Settings\All Users\
            Documents\Mes images\
            Échantillons d'images\Hiver.jpg
         </Image.Source>
      </Image>
      <TextBox TextWrapping="Wrap" Width="195"
       Margin="2,2,2,2" AcceptsReturn="True" />
      <TextBox TextWrapping="Wrap" Margin="2,2,2,2"
       AcceptsReturn="True" />
   </WrapPanel>
</Border>
```

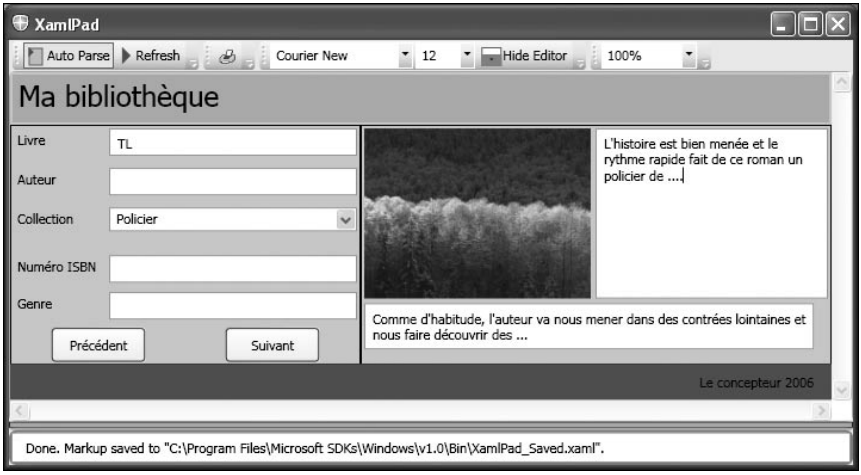
Le dernier volet est collé sur le bord gauche et contient une grille. La grille sert alors à positionner les différentes zones de saisie.

```
        <Border DockPanel.Dock="Left" MinWidth="280"
         MinHeight="200" BorderThickness="1"
         BorderBrush="Black" Background="LightGreen">
           <Grid>
```

La grille contient 2 colonnes et 6 lignes. La largeur de première colonne est fixe et plus étroite que la seconde, tandis que la hauteur de la troisième ligne est fixée à un minimum de 40 points pour que la ComboBox reste suffisamment visible.

```
              <Grid.ColumnDefinitions>
                 <ColumnDefinition Width="80" />
                 <ColumnDefinition MinWidth="200"/>
              </Grid.ColumnDefinitions>
              <Grid.RowDefinitions>
                 <RowDefinition />
                 <RowDefinition />
                 <RowDefinition MinHeight="40"/>
                 <RowDefinition />
                 <RowDefinition />
                 <RowDefinition />
              </Grid.RowDefinitions>
              <Label Name="lblLivre" Grid.Row="0"
               Grid.Column="0">
                 Livre
              </Label>
              <TextBox Name="txtLivre" Grid.Row="0"
               Grid.Column="1" MaxLength="30"
               Height="23" Text="TL"
               VerticalAlignment="Top" Margin="2,2,2,2"/>
              <Label Name="lblAuteur" Grid.Row="1"
               Grid.Column="0">
                 Auteur
              </Label>
              <TextBox Name="txtAuteur" Grid.Row="1"
               Grid.Column="1" MaxLength="30"
               CharacterCasing="Upper" Height="23"
               VerticalAlignment="Top" Margin="2,2,2,2"/>
              <Label Name="lblColl" Grid.Row="2"
               Grid.Column="0">
                 Collection
              </Label>
              <ComboBox Name="txtColl" Grid.Row="2"
               Grid.Column="1" IsEditable="True"
               VerticalAlignment="Top" Margin="2,2,2,2">
                 <ComboBoxItem>
                    Fiction
                 </ComboBoxItem>
                 <ComboBoxItem>
```

```
                    Horreur
                  </ComboBoxItem>
                  <ComboBoxItem IsSelected="True">
                    Policier
                  </ComboBoxItem>
                  <ComboBoxItem>
                    Historique
                  </ComboBoxItem>
                  <ComboBoxItem>
                    Amour
                  </ComboBoxItem>
                </ComboBox>
                <Label Name="lblISBN" Grid.Row="3"
                 Grid.Column="0">
                  Numéro ISBN
                </Label>
                <TextBox Name="txtISBN" Grid.Row="3"
                 Grid.Column="1" MaxLength="15" Height="23"
                 VerticalAlignment="Top" Margin="2,2,2,2"/>
                <Label Name="lblGenre" Grid.Row="4"
                 Grid.Column="0">
                  Genre
                </Label>
                <TextBox Name="txtGenre" Grid.Row="4"
                 Grid.Column="1" MaxLength="50"
                 Height="23" VerticalAlignment="Top"
                 Margin="2,2,2,2"/>
                <Grid Grid.Row="5" Grid.Column="0"
                 Grid.ColumnSpan="2">
                  <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                  </Grid.ColumnDefinitions>
                  <Button Grid.Row="0" Grid.Column="0"
                   Content="Précédent"
                   Width="80" Height="30" />
                  <Button Grid.Row="0" Grid.Column="1"
                   Content="Suivant"
                   Width="80" Height="30" />
                </Grid>
              </Grid>
            </Border>
          </DockPanel>
        </ScrollViewer>
</Page>
```



▲ **Figure 3-23 :** Mise en page mixte

Remarque

Erreur dans le code

Le contenu de la balise image est volontairement scindé en 3 lignes pour une question de lisibilité de l'exemple. Toutefois, si vous désirez le reproduire, il sera nécessaire de regrouper le nom du fichier sur une même ligne.

Analysons cet exemple.

La mise en page se fait au moyen d'un `DockPanel` inclus dans un `ScrollViewer`. Ce `DockPanel` est composé de 4 zones.

La première zone est le titre "Ma bibliothèque". Il s'inscrit naturellement en haut d'écran. Nous utilisons donc l'attribut `Top` pour le docking. Un cadre vert clair entoure la zone de titre. Ce cadre est réalisé avec la balise `Border`.

```
<Border BorderThickness="2" DockPanel.Dock="Top"
 BorderBrush="LightGreen">
   <Canvas Background="DarkSeaGreen" MinHeight="40" >
      <Label FontSize="24" FontFamily="Engravers MT"
       Content="Ma bibliothèque" />
   </Canvas>
</Border>
```

Comme vous le voyez, c'est le nœud `Border` qui est le parent de ce groupe. C'est donc lui qui doit être positionné par rapport à notre `DockPanel`. Les autres

éléments suivent inévitablement le mouvement. Le contrôle `Border` inclut un `Canvas`, ce qui va nous permettre de positionner notre titre sur la base de coordonnées. Grâce à l'utilisation du `Canvas`, nous pourrions avoir d'autres contrôles dans la barre de titre

Pour obtenir la couleur du fond de notre barre de titre, il nous a suffi d'utiliser l'attribut `Background` du `Canvas`. Pour éviter un écrasement exagéré, la hauteur minimale du `Canvas` est fixée à 40.

Pour obtenir le bas d'écran, nous faisons de même mais cette fois sans utiliser de bord. C'est donc directement le `Canvas` qui reçoit l'attribut `DockPanel.Dock="Bottom"`.

```
<Canvas DockPanel.Dock="Bottom" Background="Green"
 MinHeight="30">
   <Label Name="lblCopyright"
    Content="Le concepteur 2006"
    Canvas.Bottom="3" Canvas.Right="10"
    VerticalAlignment="Bottom"
    HorizontalAlignment="Right"/>
</Canvas>
```

Afin qu'ils prennent toute la largeur de la fenêtre, nous commençons par définir l'en-tête et le bas de page. Les autres zones de notre `DockPanel` sont définies après.

La troisième zone définie est la zone de droite.

```
<Border DockPanel.Dock="Right" MinWidth="200" MaxWidth="400"
 BorderThickness="1" BorderBrush="Black"
 Background="LightGreen">
   <WrapPanel>
      <Image Width="190" Margin="2,2,2,2">
         <Image.Source>
            C:\Documents and Settings\All Users\
            Documents\Mes images\
            Échantillons d'images\Hiver.jpg
         </Image.Source>
      </Image>
      <TextBox TextWrapping="Wrap" Width="195"
       Margin="2,2,2,2" AcceptsReturn="True" />
      <TextBox TextWrapping="Wrap" Margin="2,2,2,2"
       AcceptsReturn="True" />
   </WrapPanel>
</Border>
```

Dans cette zone, les contrôles sont positionnés au moyen d'un `WrapPanel`.

Il est également inclus dans un Border, simplement afin d'obtenir un fin cadre noir autour.

La dernière zone, celle de gauche, est réalisée grâce à un Grid dans lequel les différents contrôles sont positionnés dans les cellules formées par la grille. Le Grid est lui-même inclus dans un Border pour définir un fin cadre noir. Le fond est également défini dans la balise Border, mais nous aurions tout aussi bien pu le définir dans la balise Grid.

```
<Border DockPanel.Dock="Left" MinWidth="280" MinHeight="200"
 BorderThickness="1" BorderBrush="Black"
 Background="LightGreen">
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="80" />
         <ColumnDefinition MinWidth="200"/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition />
         <RowDefinition />
         <RowDefinition MinHeight="40"/>
         <RowDefinition />
         <RowDefinition />
         <RowDefinition />
      </Grid.RowDefinitions>
…
   </Grid>
</Border>
```

Si nous examinons le contenu de plus près, nous remarquons la présence d'un Grid à l'intérieur de notre Grid.

```
<Grid Grid.Row="5" Grid.Column="0"
 Grid.ColumnSpan="2">
   <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
   </Grid.ColumnDefinitions>
   <Button Grid.Row="0" Grid.Column="0"
    Content="Précédent"
    Width="80" Height="30" />
   <Button Grid.Row="0" Grid.Column="1
    Content="Suivant"
    Width="80" Height="30" />
</Grid>
```

Pourquoi ce Grid ? Tout simplement parce que les boutons doivent être centrés dans la grille et que les colonnes définies dans notre premier Grid sont de largeur différente. Ce nouveau Grid inclut, grâce à l'attribut Grid.ColumnSpan,

les deux colonnes. Dans ce nouveau `Grid`, qui occupe en définitive toute la ligne, deux colonnes, cette fois de largeur identique, sont redéfinies.

Remarque

Effet du redimensionnement

La règle du `DockPanel`, qui détermine que le dernier élément occupe l'espace restant, aura pour conséquence lorsque vous redimensionnez la fenêtre que seule la zone de gauche va s'adapter dans les deux sens. La taille des autres parties reste fixe dans une direction.

Cet exemple démontre à quel point il est possible d'intégrer les différents composants les uns dans les autres mais également que chaque choix, et ce y compris l'ordre de chaque balise, va avoir un impact sur le résultat final.

Comme les zones sont définies volontairement dans un ordre bien déterminé, les contrôles qui y sont inclus vont également être ordonnés en fonction. Cet état de fait peut éventuellement être préjudiciable à la qualité de votre interface graphique. En effet, l'ordre d'apparition des contrôles va également régir le déplacement du curseur avec la touche Tab. La solution à ce problème est simple et déjà connue de ceux qui ont l'habitude du développement traditionnel. XAML met à notre disposition l'attribut `TabIndex`, qui va simplement redéfinir l'ordre de parcours avec la touche Tab.

3.8 Créer une page composite

Une page ou une fenêtre peut être composée de différentes pages. Pour cela, vous disposez de la balise `Frame`, qui va vous permettre d'intégrer dans votre fenêtre ou votre page le contenu d'autres pages.

Prenons un exemple simple.

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Frame"
 >
   <StackPanel>
      <Label>
         Une page avec 2 frames.
      </Label>
      <WrapPanel>
         <Frame Source="Page1.xaml" BorderBrush="Black"
          BorderThickness="1" Width="200" Height="100"/>
```

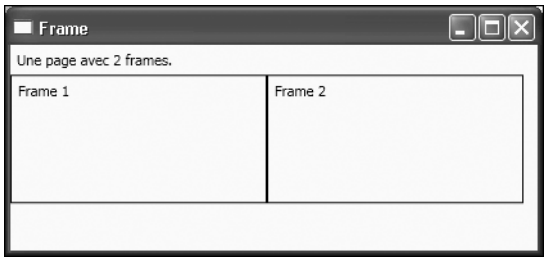
```
        <Frame Source="Page2.xaml" BorderBrush="Black"
         BorderThickness="1" Width="200" Height="100"/>
      </WrapPanel>
   </StackPanel>
</Window>
```

Page1.xaml contient :

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <StackPanel>
      <Label>
         Frame 1
      </Label>
   </StackPanel>
</Page>
```

Page2.xaml contient :

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <StackPanel>
      <Label>
         Frame 2
      </Label>
   </StackPanel>
</Page>
```



◄ **Figure 3-24 :** Utilisation de frames

Attention

Contenu d'un frame

Si vous pouvez utiliser des frames aussi bien au sein d'une balise `Window` que d'une balise `Page`, la page référencée par la balise `Frame` ne pourra en aucun cas être une page utilisant la balise `Window`.

3.9 Checklist

Dans ce chapitre, nous avons essentiellement vu comment disposer les éléments à l'écran et principalement :

- le positionnement par utilisation des coordonnées en utilisant `Canvas` ;
- le positionnement en utilisant une grille avec la classe `Grid` ;
- le positionnement en utilisant les panneaux de type `StackPanel`, `WrapPanel` et `DockPanel` ;
- le mélange de ces différentes méthodes ;
- autoriser le défilement avec `ScrollViewer` ;
- réaliser des pages composées de plusieurs pages avec `Frame` ;
- comprendre et utiliser les attributs attachés.

Les autres contrôles de base

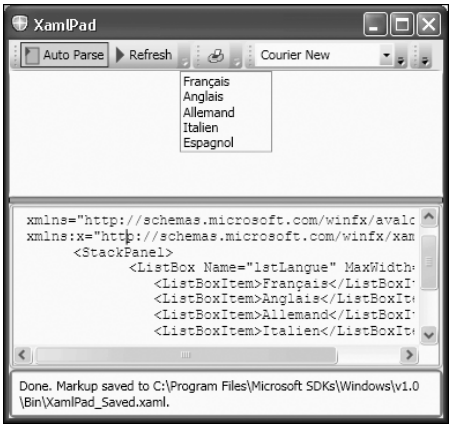
Créer une liste déroulante 92
Créer une ComboBox 98
Créer une case à cocher 100
Utiliser les boutons radio 102
Placer des info-bulles 106
Utiliser les panneaux à onglets 109
Créer un bouton automatique 112
Utiliser un Slider 114
Utiliser un Expander 118
Utiliser une ViewBox 121
Utiliser un Popup 123
Ajouter de la vidéo dans la fenêtre 126
Checklist 129

4.1 Créer une liste déroulante

Pour faire un choix parmi une liste, vous pouvez utiliser ce contrôle. Toutefois, la liste ne doit pas être trop longue sinon le choix devient vite fastidieux.

Pour réaliser une liste de choix, vous devez utiliser une balise `ListBox`. À l'intérieur du nœud ainsi défini, vous devrez pour chaque élément de la liste ajouter un nœud enfant utilisant la balise `ListBoxItem`.

```
<ListBox Name="lstLangue" MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem>Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```

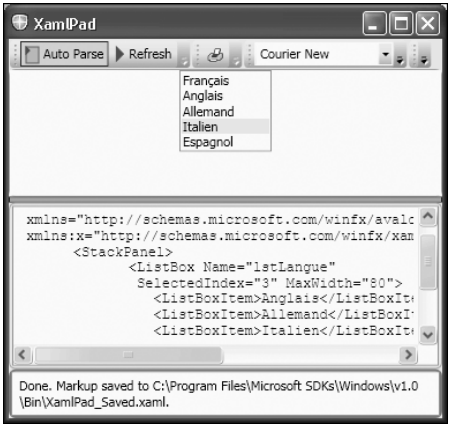


◄ **Figure 4-1** : Une simple liste de choix

Outre les attributs que nous avons déjà évoqués comme `MaxWidth`, repris dans l'exemple, deux attributs sont particulièrement intéressants pour ce contrôle.

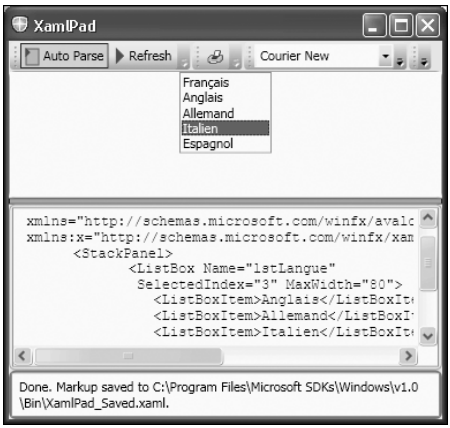
Il s'agit tout d'abord de `SelectedIndex`, qui permet de définir une valeur par défaut.

```
<ListBox Name="lstLangue"
   SelectedIndex="3" MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem>Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```



◄ **Figure 4-2** : La présélection dans une liste de choix

Comme vous pouvez le constater en comparant le résultat avec l'écran ci-dessous, la présélection ne s'affiche pas de la même manière qu'un élément sélectionné par l'utilisateur.



◄ **Figure 4-3** : Un élément sélectionné dans une liste de choix

Attention

Particularité de la numérotation

Les différentes lignes de la liste de choix sont numérotées non pas de 1 à n mais bien de 0 à n-1 ; n étant le nombre d'éléments de la liste. Pour sélectionner le premier, vous devez donc mettre comme valeur 0. Dans l'exemple, la valeur sélectionnée est bien la quatrième, `SelectedIndex` vaut 3.

Il existe une autre façon pour indiquer quel élément doit être présélectionné. Cette méthode a l'avantage d'être beaucoup plus visuelle car il s'agit de modifier dans la balise de l'élément voulu l'attribut `IsSelected`. Le résultat obtenu est le même qu'avec l'attribut `SelectedIndex`.

```
<ListBox Name="lstLangue"
   MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem IsSelected="true">Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```

Le second attribut intéressant est l'attribut `SelectionMode`, qui permet d'autoriser ou d'interdire la sélection de plusieurs valeurs. Si vous optez pour plusieurs valeurs, il existe deux modes possibles, en assignant soit la valeur `Multiple`, ce qui permet de sélectionner ou de désélectionner plusieurs valeurs en cliquant simplement avec la souris, soit la valeur `Extend`, ce qui permet d'utiliser les touches Ctrl et Maj pour réaliser votre sélection. La touche Ctrl permet d'ajouter à votre sélection l'élément sur lequel vous cliquez sans désélectionner les éléments déjà choisis. La touche Maj permet quant à elle de sélectionner tous les éléments repris entre le dernier élément sélectionné et l'élément sur lequel vous cliquez.

```
<ListBox Name="lstLangue"
   SelectionMode="Multiple" MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem>Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```

◄ **Figure 4-4 :** Une liste de choix à sélection multiple

Si vous désirez réaliser une présélection de plusieurs valeurs, il suffit de modifier l'attribut `IsSelected` de chaque élément à présélectionner.

```
<ListBox Name="lstLangue"
   MaxWidth="80" SelectionMode="Multiple">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem IsSelected="true">Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem IsSelected="true">Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```

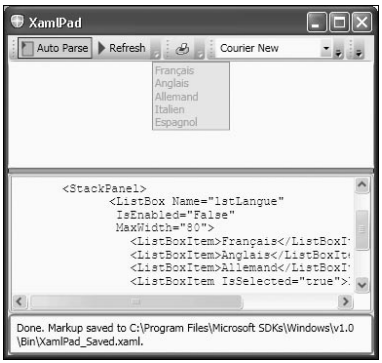
Remarque

Présélectionner un élément avec `SelectedValue`

Au lieu d'utiliser `SelectedIndex`, vous pourriez utiliser l'attribut `SelectedValue`. Toutefois, dans le code XAML, cela me semble peu opportun et plus compliqué. Il n'en va pas nécessairement de même si vous initialisez la valeur depuis le code .NET.

Si vous désirez rendre une liste inaccessible à l'utilisateur, vous devrez utiliser l'attribut `IsEnabled`. Ce contrôle ne supporte pas l'attribut `IsReadOnly`.

```
<ListBox Name="lstLangue"
  IsEnabled="False"
   MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem IsSelected="true">Italien</ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```



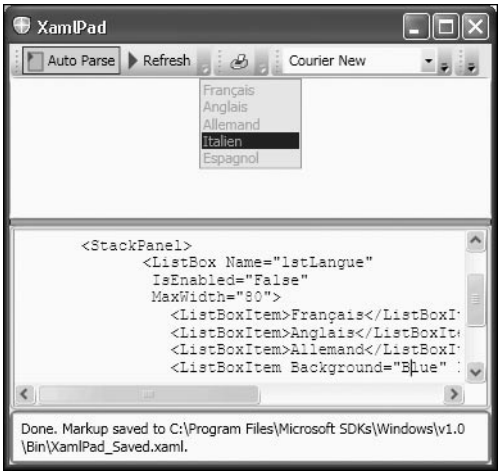
◄ **Figure 4-5 :** Liste de choix désactivée

Astuce

Problème de lisibilité

Cette façon de procéder ne convient pas bien car, comme vous pouvez le constater, bien qu'une valeur soit présélectionnée, elle n'est pas visible. Pour résoudre ce problème, assignez la valeur `Blue` à l'attribut `Background` de la balise `ListBoxItem` correspondante.

```
<ListBox Name="lstLangue"
   MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem Background="Blue" IsSelected="true">
      Italien
   </ListBoxItem>
   <ListBoxItem>Espagnol</ListBoxItem>
</ListBox>
```



◄ **Figure 4-6 :** Liste de choix désactivée modifiée

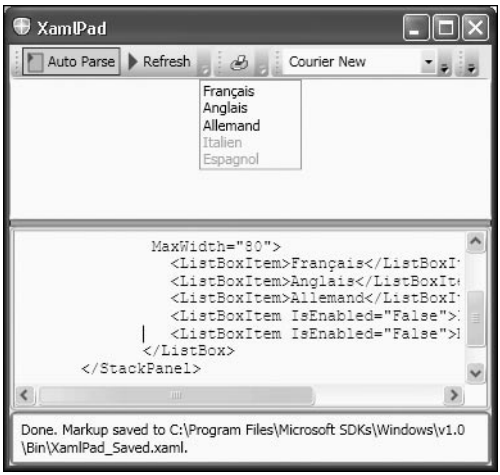
Attention

Limite de cette modification

Cette solution n'est envisageable que si le contrôle reste toujours à l'état inactif. Dans le cas contraire, l'élément modifié avec `Background` apparaîtra toujours comme sélectionné.

Si vous souhaitez que le contrôle soit actif mais que certaines valeurs soient désactivées, il suffit de placer l'attribut `IsEnabled` dans l'état `False` pour chacun des éléments.

```
<ListBox Name="lstLangue"
   MaxWidth="80">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem IsEnabled="False">Italien</ListBoxItem>
   <ListBoxItem IsEnabled="False">Espagnol</ListBoxItem>
</ListBox>
```



◄ **Figure 4-7 :** Une liste de choix dont certains éléments sont désactivés

Si la hauteur de la liste n'est pas suffisante, une barre de défilement sera automatiquement ajoutée.

```
<ListBox Name="lstLangue"
   MaxWidth="80"  MaxHeight="45">
   <ListBoxItem>Français</ListBoxItem>
   <ListBoxItem>Anglais</ListBoxItem>
   <ListBoxItem>Allemand</ListBoxItem>
   <ListBoxItem IsEnabled="False">Italien</ListBoxItem>
   <ListBoxItem IsEnabled="False">Espagnol</ListBoxItem>
</ListBox>
```

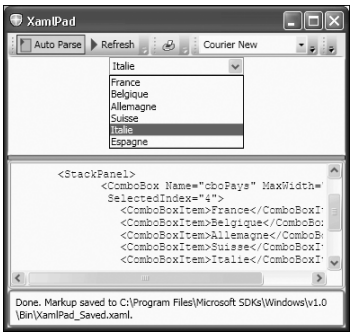
◄ **Figure 4-8 :** Une liste avec sa barre de défilement

4.2 Créer une ComboBox

Très proche de la liste de choix (**ListBox**), la **ComboBox** a toutefois généralement ma préférence. Elle occupe moins de place à l'écran et permet de rechercher un élément en tapant le début de la valeur. Elle offre également la possibilité d'introduire une valeur différente de celles proposées dans la liste mais uniquement si vous désirez offrir cette possibilité.

Pour créer une **ComboBox**, vous devrez utiliser la balise du même nom. À l'intérieur du nœud ainsi créé, vous devrez pour chaque élément créer un nœud enfant en utilisant la balise `ComboBoxItem`.

```
<ComboBox Name="cboPays" MaxWidth="80"
 SelectedIndex="4">
   <ComboBoxItem>France</ComboBoxItem>
   <ComboBoxItem>Belgique</ComboBoxItem>
   <ComboBoxItem>Allemagne</ComboBoxItem>
   <ComboBoxItem>Suisse</ComboBoxItem>
   <ComboBoxItem>Italie</ComboBoxItem>
   <ComboBoxItem>Espagne</ComboBoxItem>
</ComboBox>
```

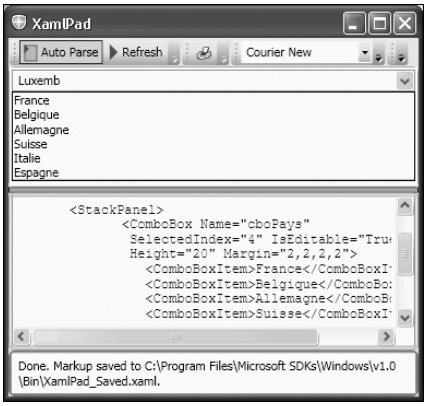


◄ **Figure 4-9 :** Une simple ComboBox

Renvoi *Comme pour le contrôle* `ListBox`*, l'attribut* `SelectedIndex` *permet de sélectionner une valeur par défaut. Voir page 92.*

L'attribut `IsEditable` permet de définir si oui ou non vous pouvez encoder une valeur différente d'une des valeurs de la liste.

```
<ComboBox Name="cboPays"
 SelectedIndex="4" IsEditable="True"
 Height="20" Margin="2,2,2,2">
   <ComboBoxItem>France</ComboBoxItem>
   <ComboBoxItem>Belgique</ComboBoxItem>
   <ComboBoxItem>Allemagne</ComboBoxItem>
   <ComboBoxItem>Suisse</ComboBoxItem>
   <ComboBoxItem>Italie</ComboBoxItem>
   <ComboBoxItem>Espagne</ComboBoxItem>
</ComboBox>
```



◀ **Figure 4-10 :** Une ComboBox éditable

Le fait que ce contrôle soit éditable a pour conséquence qu'il possède à la fois des attributs de `ListBox` et de `TextBox`. C'est pourquoi vous pouvez par exemple assigner l'attribut `Text`, qui aura pour effet de placer cette valeur dans la zone de saisie.

Attention

L'attribut IsReadOnly

L'attribut `IsReadOnly` provoque un comportement inattendu. En effet, le fait de placer le contrôle en lecture seule au moyen de cet attribut n'empêche pas l'utilisateur de choisir dans la liste et ainsi de changer la valeur. Cet attribut n'affecte que la zone de saisie du texte. Si vous désirez que la valeur ne puisse être changée, il est nécessaire d'assigner la valeur `False` à l'attribut `IsEnabled`.

Si pour une raison particulière vous désirez retirer la possibilité de rechercher dans la liste en tapant successivement les lettres du début du mot recherché, l'attribut `IsTextSearchEnabled` doit être mis à `False`.

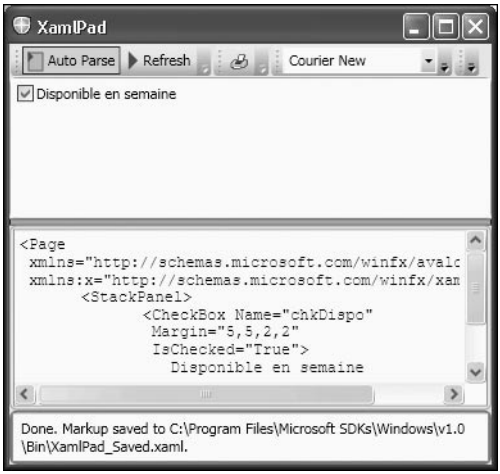
```
<ComboBox Name="cboPays"
 SelectedIndex="4" IsTextSearchEnabled="False"
 Height="20" Margin="2,2,2,2">
   <ComboBoxItem>France</ComboBoxItem>
   <ComboBoxItem>Belgique</ComboBoxItem>
   <ComboBoxItem>Allemagne</ComboBoxItem>
   <ComboBoxItem>Suisse</ComboBoxItem>
   <ComboBoxItem>Italie</ComboBoxItem>
   <ComboBoxItem>Espagne</ComboBoxItem>
</ComboBox>
```

Toutefois, il est préférable de ne pas changer cette option.

4.3 Créer une case à cocher

La case à cocher fait partie de la panoplie des contrôles indispensables pour réaliser une interface graphique de qualité. Comme tous les contrôles vus précédemment, il s'utilise très facilement. Il suffit d'utiliser la balise `CheckBox`.

```
<CheckBox Name="chkDispo"
 Margin="5,5,2,2"
 IsChecked="True">
   Disponible en semaine
</CheckBox>
```

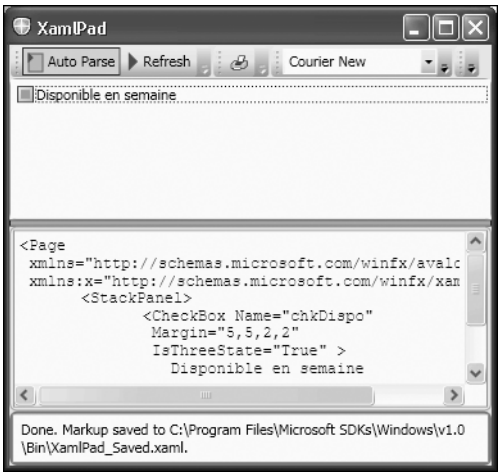


◄ **Figure 4-11 :** Une case à cocher

L'attribut `IsChecked` reçoit une valeur booléenne qui indique si par défaut la case est cochée ou non. Si vous ne souhaitez pas que la case soit cochée par défaut, vous pouvez bien sûr omettre tout simplement cet attribut.

Parfois, vous aurez besoin d'une case à cocher autorisant l'état indéterminé. C'est l'attribut `IsThreeState` qui va autoriser ce comportement.

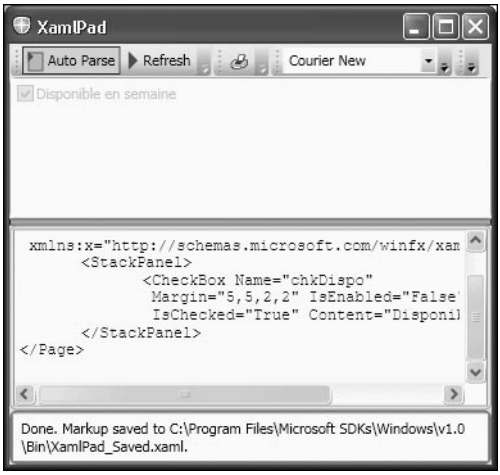
```
<CheckBox Name="chkDispo"
 Margin="5,5,2,2"
 IsThreeState="True" >
   Disponible en semaine
</CheckBox>
```



◄ **Figure 4-12 :** Une case à cocher dans l'état indéterminé

Si vous souhaitez utiliser ce contrôle uniquement pour afficher une information mais que vous ne souhaitiez pas que l'utilisateur puisse modifier l'état de la case à cocher, vous devez ici encore utiliser obligatoirement l'attribut `IsEnabled`.

```
<CheckBox Name="chkDispo"
 Margin="5,5,2,2" IsEnabled="False"
 IsChecked="True" Content="Disponible en semaine" />
```



◄ **Figure 4-13** : Une case à cocher désactivée

Remarque

L'attribut Content

Dans ce dernier exemple, le texte est assigné à l'attribut `Content` au lieu de le placer dans le nœud. Le résultat est identique, c'est une question de goût personnel.

4.4 Utiliser les boutons radio

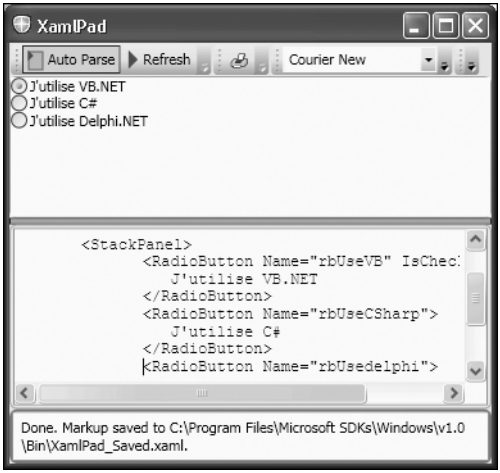
Le contrôle `RadioButton` est un autre moyen de faire un choix dans une liste. La syntaxe pour ajouter un bouton radio est fort simple.

```
<RadioButton Name="rbUse" IsChecked="True">
   J'utilise XAML
</RadioButton>
```

Comme pour la case à cocher, nous retrouvons l'attribut `IsChecked`. À la différence de la case à cocher, les boutons radio sont associés les uns aux autres. Ce qui fait que, quand vous sélectionnez un bouton radio, les autres sont automatiquement désélectionnés.

```
<RadioButton Name="rbUseVB" IsChecked="True">
   J'utilise VB.NET
</RadioButton>
<RadioButton Name="rbUseCSharp">
```

```
   J'utilise C#
</RadioButton>
<RadioButton Name="rbUsedelphi">
   J'utilise Delphi.NET
</RadioButton>
```



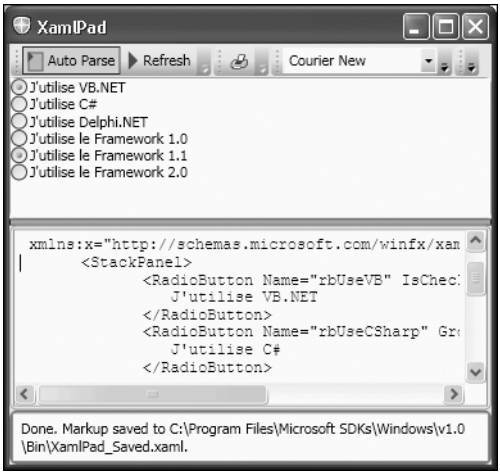
◄ **Figure 4-14 :** Utiliser des boutons radio

Si vous désirez avoir dans un même écran plusieurs listes de boutons radio indépendantes les unes des autres, vous devez les intégrer dans un ensemble. Dans les versions précédentes, nous aurions dû utiliser une `RadioButtonList` mais elle n'est actuellement plus disponible en XAML.

Pour regrouper des boutons radio dans des ensembles différents, le moyen le plus simple est de leur ajouter un attribut `GroupName`. Assignez la même valeur à cet attribut pour tous les boutons radio devant être associés.

```
<RadioButton Name="rbUseVB" IsChecked="True"
 GroupName="grpLanguage">
   J'utilise VB.NET
</RadioButton>
<RadioButton Name="rbUseCSharp" GroupName="grpLanguage">
   J'utilise C#
</RadioButton>
<RadioButton Name="rbUsedelphi" GroupName="grpLanguage">
   J'utilise Delphi.NET
</RadioButton>
<RadioButton Name="rbUse10" GroupName="grpFramework">
   J'utilise le Framework 1.0
</RadioButton>
<RadioButton Name="rbUse11" IsChecked="True"
```

```
 GroupName="grpFramework">
   J'utilise le Framework 1.1
</RadioButton>
<RadioButton Name="rbUse20" GroupName="grpFramework">
   J'utilise le Framework 2.0
</RadioButton>
```



◄ **Figure 4-15 :** Utiliser des boutons radio

Une autre solution consiste à utiliser un contrôle conteneur séparé pour chaque liste de boutons radio. Dans l'exemple ci-dessous, nous utiliserons deux `StackPanel` supplémentaires. C'est pourquoi, pour cet exemple, le code complet vous est à nouveau présenté.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <StackPanel>
      <StackPanel>
         <RadioButton Name="rbUseVB" IsChecked="True">
            J'utilise VB.NET
         </RadioButton>
         <RadioButton Name="rbUseCSharp">
            J'utilise C#
         </RadioButton>
         <RadioButton Name="rbUsedelphi">
            J'utilise Delphi.NET
         </RadioButton>
      </StackPanel>
      <StackPanel>
```

```
        <RadioButton Name="rbUse10">
           J'utilise le Framework 1.0
        </RadioButton>
        <RadioButton Name="rbUse11" IsChecked="True">
           J'utilise le Framework 1.1
        </RadioButton>
        <RadioButton Name="rbUse20">
           J'utilise le Framework 2.0
        </RadioButton>
      </StackPanel>
   </StackPanel>
</Page>
```

Le résultat est identique à la méthode précédente. Toutefois, l'utilisation d'une méthode ou d'une autre peut influencer la mise en page.

Astuce

Liste de choix non modifiable

Pour rendre une liste de choix non modifiable, vous devez utiliser l'attribut `IsEnabled`. Celui-ci doit être appliqué sur chaque bouton radio. Toutefois, si elle est incluse dans un conteneur qui lui est spécifique, vous pouvez spécifier l'attribut `IsEnabled` dans le conteneur. Cette façon de faire est bien plus pratique à bien des égards.

Normalement, il devrait être possible d'utiliser un contrôle de type `GroupBox` au lieu d'un `Canvas`. Toutefois, dans la version bêta utilisée au moment d'écrire ces lignes, cette possibilité n'était pas supportée. Le contrôle `GroupBox` existe bel et bien mais ne supporte qu'un enfant et ne permet dès lors pas de regrouper les boutons radio. Son utilisation se limite à l'affichage du traditionnel contour.

Son utilité reste malgré tout évidente pour rendre votre interface claire et bien compréhensible pour l'utilisateur.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <StackPanel>
      <GroupBox BorderThickness="1" BorderBrush="Black"
       Header="Language" Width="150">
        <StackPanel>
           <RadioButton Name="rbUseVB" IsChecked="True">
              J'utilise VB.NET
           </RadioButton>
           <RadioButton Name="rbUseCSharp">
              J'utilise C#
           </RadioButton>
```

```
            <RadioButton Name="rbUsedelphi">
               J'utilise Delphi.NET
            </RadioButton>
         </StackPanel>
      </GroupBox>
      <GroupBox BorderThickness="1" BorderBrush="Black"
       Header="Framework" Width="150">
         <StackPanel>
            <RadioButton Name="rbUse10">
               J'utilise le Framework 1.0
            </RadioButton>
            <RadioButton Name="rbUse11" IsChecked="True">
               J'utilise le Framework 1.1
            </RadioButton>
            <RadioButton Name="rbUse20">
               J'utilise le Framework 2.0
            </RadioButton>
         </StackPanel>
      </GroupBox>
   </StackPanel>
</Page>
```

◄ **Figure 4-16 :** Utilisation d'un GroupBox

Notez au passage l'utilisation de l'attribut `Header` pour indiquer le titre de votre `GroupBox`.

4.5 Placer des info-bulles

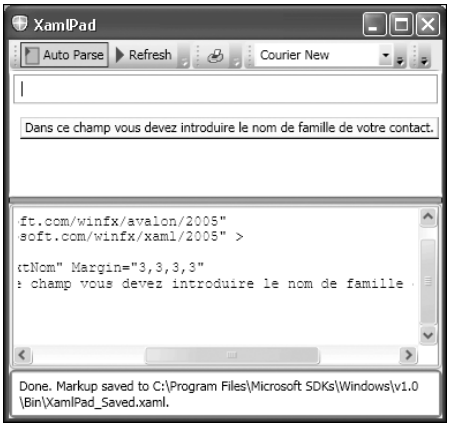
Bien que, contrairement aux contrôles vus précédemment, la bulle d'information ne puisse exister sans un autre contrôle, une place privilégiée a été réservée à cette fonctionnalité car les bulles d'information sont souvent trop peu utilisées

en dehors des programmes commerciaux. Pourtant, elles apportent un réel plus à votre développement et, surtout, elles évitent pas mal d'incompréhension et de confusion pour l'utilisateur de votre interface. Pour des questions de place à l'écran, les étiquettes précédant les différents champs de saisie ou de choix sont généralement fortement résumées et très peu explicites. Les bulles d'information sont là pour pallier ce manque et constituent le premier niveau d'aide.

Pour réaliser une telle bulle, vous devez utiliser l'attribut `ToolTip` du contrôle auquel la bulle d'information doit être associée.

À titre d'exemple, nous allons associer une bulle d'information à une boîte de saisie de texte.

```
<TextBox Name="txtNom" Margin="3,3,3,3"
 ToolTip="Dans ce champ vous devez introduire
          le nom de famille de votre contact." />
```

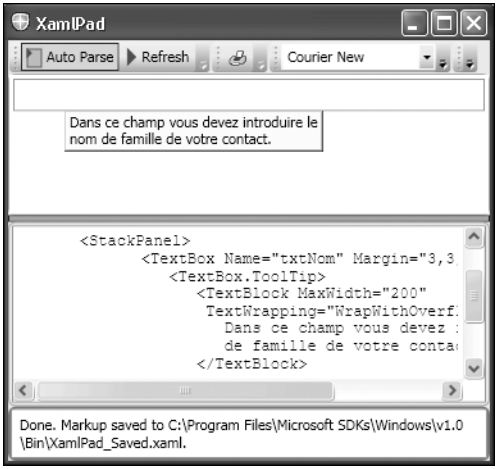


◄ **Figure 4-17 :** Une info-bulle

Cette façon de faire est toutefois limitée à la présentation sur une seule ligne. Pour améliorer notre info-bulle, nous devons utiliser une syntaxe légèrement plus compliquée en la définissant comme un nœud fils.

```
<TextBox Name="txtNom" Margin="3,3,3,3">
   <TextBox.ToolTip>
      <TextBlock MaxWidth="200"
       TextWrapping="WrapWithOverflow" >
         Dans ce champ vous devez introduire le nom
         de famille de votre contact.
      </TextBlock>
   </TextBox.ToolTip>
</TextBox>
```

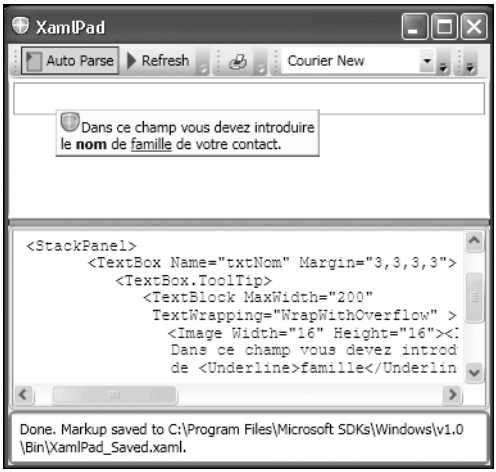
De cette façon, le texte d'information est alors inclus dans un bloc de texte, ce qui permet d'ajuster les attributs d'affichage pour obtenir le résultat souhaité. Dans l'exemple, la taille de la bulle est limitée à 150 pixels et le texte passe automatiquement à la ligne en étendant la zone autant que nécessaire.



◄ **Figure 4-18 :** Une info-bulle sur plusieurs lignes

Outre le passage à la ligne, cette façon d'aborder le problème offre également la possibilité d'enrichir la présentation du contenu.

```
<TextBox Name="txtNom" Margin="3,3,3,3">
   <TextBox.ToolTip>
      <TextBlock MaxWidth="200"
       TextWrapping="WrapWithOverflow" >
         <Image Width="16" Height="16">
            <Image.Source>
               C:\Windows\Microsoft.NET\Windows\
               v6.0.5070\Avalon\avalonArp.ico
            </Image.Source>
         </Image>
         Dans ce champ vous devez introduire
         le <Bold>nom</Bold> de <Underline>famille
         </Underline> de votre contact.
      </TextBlock>
   </TextBox.ToolTip>
</TextBox>
```



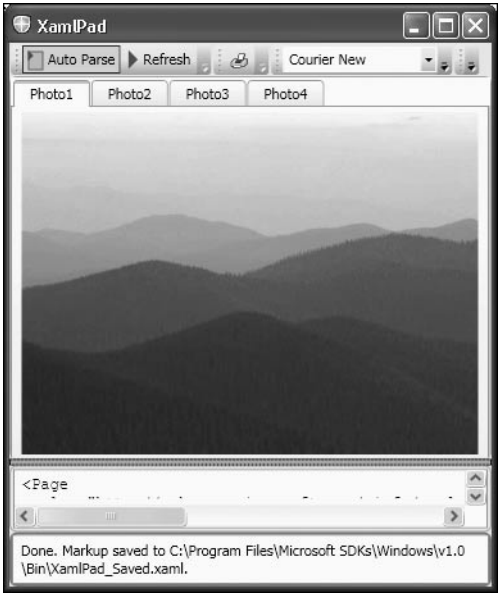
◄ **Figure 4-19 :** Gras et italique dans une info-bulle

4.6 Utiliser les panneaux à onglets

Une dernière façon de réaliser une mise en page est d'utiliser un panneau à onglets.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <TabControl MinWidth="300" MinHeight="300">
      <TabItem Header="Photo1">
         <Image Source="C:\Documents and Settings\
                   All Users\Documents\Mes images\
                   Échantillons d'images\Collines.jpg" />
      </TabItem>
      <TabItem Header="Photo2">
         <Image Source="C:\Documents and Settings\
                   All Users\Documents\Mes images\
                   Échantillons d'images\
                   Coucher de soleil.jpg" />
      </TabItem>
      <TabItem Header="Photo3">
         <Image Source="C:\Documents and Settings\
                   All Users\Documents\Mes images\
                   Échantillons d'images\Hiver.jpg" />
      </TabItem>
      <TabItem Header="Photo4">
         <Image Source="C:\Documents and Settings\
                   All Users\Documents\Mes images\
```

```
                    Échantillons d'images\Nénuphars.jpg" />
      </TabItem>
   </TabControl>
</Page>
```



◄ **Figure 4-20 :** Utilisation des onglets

Bien qu'il représente une surface, le `TabControl` est plus proche des contrôles de type `TextBox` que du `Canvas`, par exemple. En effet, chaque nœud `TabItem` ne peut avoir qu'un seul enfant.

***Pour pallier ce problème, la solution est très simple et revient à appliquer les règles qui sont vues dans le chapitre* Mélanger les techniques de mise en page** *(voir page 81).*

Ajoutez ce `TabItem` à notre code précédent.

```
<TabItem Header="Miniatures" IsSelected="True">
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition/>
         <RowDefinition/>
      </Grid.RowDefinitions>
```

```
      <Image Grid.Column="0" Grid.Row="0"
       Source="C:\Documents and Settings\All Users\
        Documents\Mes images\Échantillons d'images\
        Collines.jpg" />
      <Image Grid.Column="0" Grid.Row="1"
       Source="C:\Documents and Settings\All Users\
        Documents\Mes images\Échantillons d'images\
        Coucher de soleil.jpg" />
      <Image Grid.Column="1" Grid.Row="0"
       Source="C:\Documents and Settings\All Users\
        Documents\Mes images\Échantillons d'images\
        Hiver.jpg" />
      <Image Grid.Column="1" Grid.Row="1"
       Source="C:\Documents and Settings\All Users\
        Documents\Mes images\Échantillons d'images\
        Nénuphars.jpg" />
   </Grid>
</TabItem>
```

◄ **Figure 4-21 :**
Utilisation des onglets

Remarque

Utilisation de IsSelected

Remarquez au passage l'utilisation de `IsSelected` pour spécifier un onglet par défaut différent du premier.

4.7 Créer un bouton automatique

Nous avons déjà vu le bouton classique précédemment mais, dans certaines circonstances, son utilisation peut se révéler peu pratique. Prenons un exemple : si vous désirez utiliser un bouton pour faire défiler des données dans un écran, avec un bouton traditionnel vous devrez cliquer de manière répétée. Pour éviter cela, vous disposez d'un bouton particulier appelé `RepeatButton` qui va répéter automatiquement le clic tant que vous maintiendrez la pression sur le bouton.

```
<RepeatButton Name="btnSuivant"
 Width="80" Height="30">
   Suivant
</RepeatButton>
```

Pour obtenir un contrôle efficace du comportement de ce bouton à répétition, nous disposons tout d'abord de l'attribut `Interval`, qui détermine le temps entre deux appels de l'action associée au bouton.

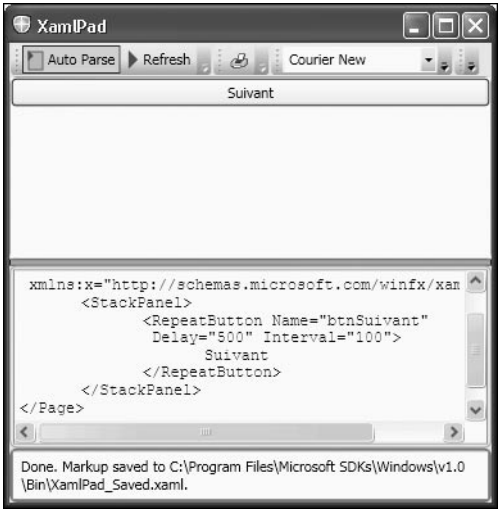
```
<RepeatButton Name="btnSuivant"
 Width="80" Height="30"
 Interval="100">
   Suivant
</RepeatButton>
```

Dans l'exemple, l'opération *Suivant* sera effectuée tous les dixièmes de seconde.

Pour éviter que la répétition ne démarre directement, vous pouvez imposer un délai. Si la pression sur le bouton est inférieure au délai, il se comportera comme un bouton normal. En revanche, si la pression se prolonge au-delà du délai fixé, la répétition de tâche va pouvoir démarrer. Pour contrôler ce délai, vous disposez de l'attribut `Delay`.

```
<RepeatButton Name="btnSuivant"
 Delay="500" Interval="100">
   Suivant
</RepeatButton>
```

Dans cet exemple, le délai est fixé à une demi-seconde (500 millièmes).



◄ **Figure 4-22 :** Le bouton à répétition

Attention

Durée du traitement

La durée du traitement à réaliser à chaque répétition ne peut excéder l'intervalle prévu entre chaque répétition. Dans le cas contraire, le traitement pourrait se poursuivre encore un certain temps après que l'utilisateur eut relâché le bouton.

Comme vous pouvez le constater, visuellement, il n'y a aucune différence entre les deux types de boutons.

Si vous désirez visuellement identifier ce type de bouton, vous pouvez adapter pour lui votre charte graphique en changeant par exemple la couleur du fond. Pour rappel, vous disposez pour cela de l'attribut `Background`. Vous pouvez également jouer sur l'effet visuel lié à la forme du curseur de la souris.

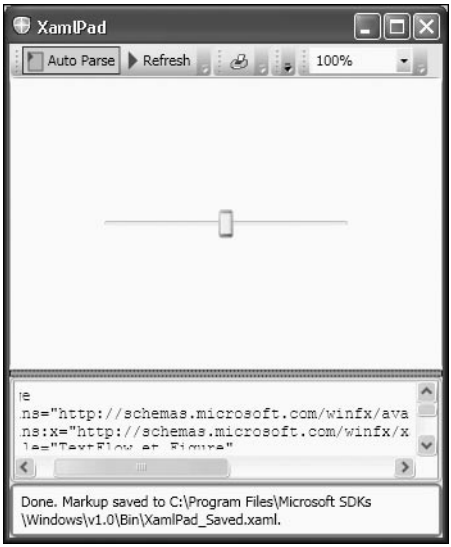
```
<RepeatButton Name="btnSuivant"
 Delay="500" Interval="100"
 Cursor="Hand">
   Suivant
</RepeatButton>
```

Dans l'exemple ci-dessus, le curseur de la souris est transformé en une main lors du passage sur le bouton.

4.8 Utiliser un Slider

Le `Slider` offre un moyen très visuel pour introduire une valeur numérique. Il est souvent utilisé lorsque la valeur a une action directe sur l'interface utilisateur, comme un zoom. Ce contrôle dispose en plus des attributs courants, d'un nombre important d'attributs permettant de lui donner le comportement voulu. Commençons par un `Slider` simple permettant de définir une valeur entre 0 et 100.

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"/>
```



◄ **Figure 4-23 :** Utilisation d'un Slider

Généralement, ce genre de contrôle dispose de repères visuels. Il existe deux possibilités pour les ajouter, en utilisant soit l'attribut `TickFrequency`, qui vous permet de définir l'écart entre chaque repère :

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 TickFrequency="10" TickPlacement="BottomRight"/>
```

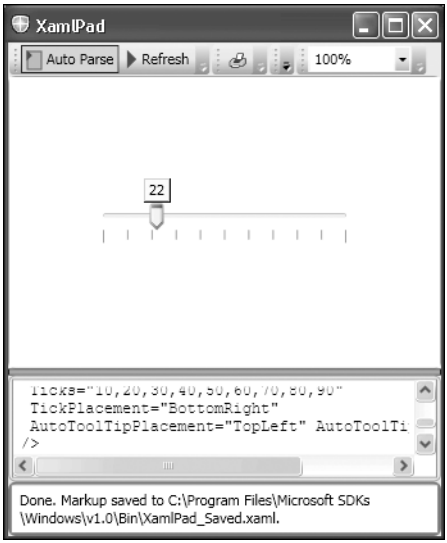
soit l'attribut `Ticks` :

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"/>
```

Ce dernier est plus contraignant mais permet d'avoir des repères qui sont irréguliers.

En plus des repères visuels, vous pouvez ajouter automatiquement la valeur dans un `ToolTip`.

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"
 AutoToolTipPlacement="TopLeft" AutoToolTipPrecision="0"
/>
```



◄ **Figure 4-24 :** Un Slider avec repères visuels

Astuce

Limiter les valeurs aux repères

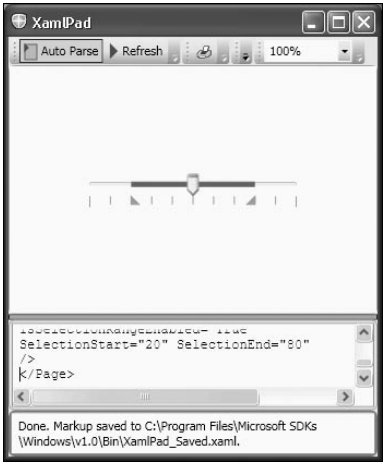
Avec l'attribut `IsSnapToTickEnabled`, les valeurs sont automatiquement arrondies au repère le plus proche.

Il est également possible de modifier la valeur sans déplacer le curseur mais en cliquant sur la barre. `LargeChange` permet de définir la valeur qui sera ajoutée ou retirée automatiquement à chaque clic. Les attributs `Delay` et `Interval` permettent de moduler automatiquement la répétition de l'action.

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"
 AutoToolTipPlacement="TopLeft" AutoToolTipPrecision="0"
 LargeChange="10" Delay="500" Interval="200"
/>
```

Vous pouvez également définir une zone visuelle où devrait se trouver la valeur. Cette zone est un attribut purement visuel mais ne limite en rien les valeurs possibles.

```
<Slider Width="200" VerticalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"
 AutoToolTipPlacement="TopLeft" AutoToolTipPrecision="0"
 LargeChange="10" Delay="500" Interval="200"
 IsSelectionRangeEnabled="True"
 SelectionStart="20" SelectionEnd="80"
/>
```

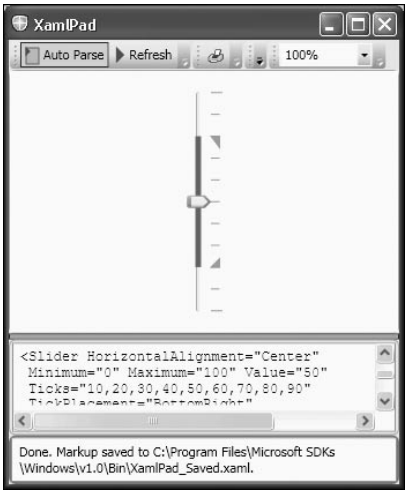


◄ **Figure 4-25 :** Un Slider avec une zone

Si vous préférez un `Slider` vertical, aucun problème, la propriété `Orientation` vous permet de choisir entre horizontal (par défaut) ou vertical.

```
<Slider HorizontalAlignment="Center"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"
 AutoToolTipPlacement="TopLeft" AutoToolTipPrecision="0"
```

```
 LargeChange="10" Delay="500" Interval="200"
 IsSelectionRangeEnabled="True"
 SelectionStart="20" SelectionEnd="80"
 Height="200" Orientation="Vertical"
/>
```



◄ **Figure 4-26 :** Un Slider vertical

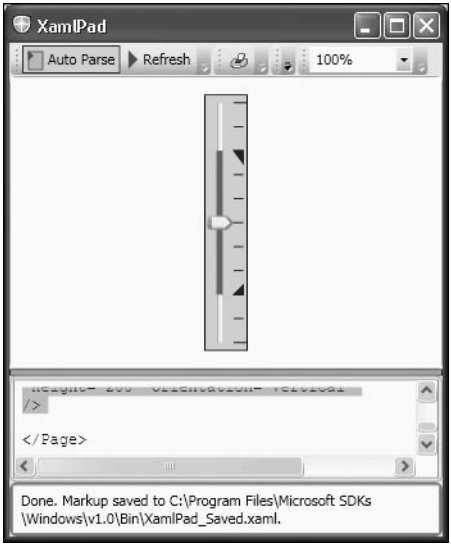
Attention

Définition de la taille

En orientation verticale, la taille est contrôlée non plus par l'attribut `Width` mais bien par l'attribut `Height`.

Bien sûr, ce contrôle accepte les attributs habituels.

```
<Slider HorizontalAlignment="Center"
 BorderBrush="Black" BorderThickness="1"
 Foreground="Blue" Background="LightGray"
 Minimum="0" Maximum="100" Value="50"
 Ticks="10,20,30,40,50,60,70,80,90"
 TickPlacement="BottomRight"
 AutoToolTipPlacement="TopLeft" AutoToolTipPrecision="0"
 LargeChange="10" Delay="500" Interval="200"
 IsSelectionRangeEnabled="True"
 SelectionStart="20" SelectionEnd="80"
 Height="200" Orientation="Vertical"
/>
```



◄ **Figure 4-27 :** Un Slider coloré

4.9 Utiliser un Expander

Si vous avez beaucoup d'informations à afficher sur votre écran, il peut rapidement devenir surchargé et peu lisible. Pour remédier à cela, vous pouvez comme nous l'avons vu précédemment utiliser les onglets.

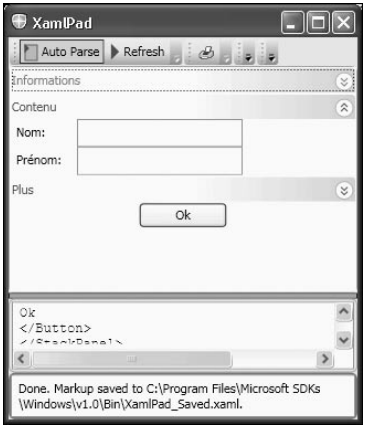
Pour plus d'informations sur l'utilisation des onglets, voyez le chapitre ***Utiliser les panneaux à onglets*** *page 109.*

Une autre solution est d'utiliser un `Expander`. Celui-ci a la faculté d'afficher ou de cacher son contenu en ne laissant alors visible que son titre.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Expander"
 >
   <StackPanel>
      <Expander Header="Informations" IsExpanded="false">
         <TextBlock>
            Cette zone contient un texte d'information
         </TextBlock>
      </Expander>
```

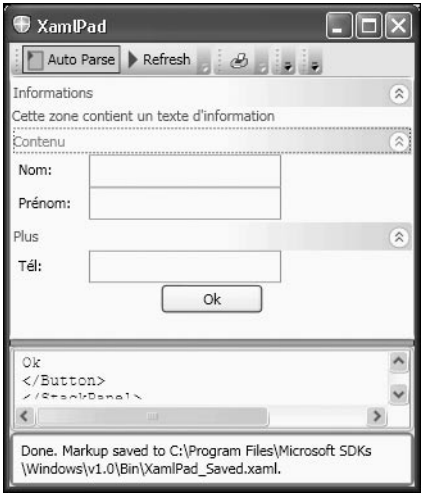
Un Expander peut parfaitement contenir plus d'un contrôle utilisateur mais, dans ce cas, ils devront être encapsulés dans un conteneur.

```
      <Expander Header="Contenu" IsExpanded="true">
         <StackPanel>
            <WrapPanel>
               <Label Width="60">
                  Nom:
               </Label>
               <TextBox Name="txtNom" Width="150"/>
            </WrapPanel>
            <WrapPanel>
               <Label Width="60">
                  Prénom:
               </Label>
               <TextBox Name="txtprenom" Width="150"/>
            </WrapPanel>
         </StackPanel>
      </Expander>
      <Expander Header="Plus">
         <WrapPanel>
            <Label Width="60">
               Tél:
            </Label>
            <TextBox Name="txttel" Width="150"/>
         </WrapPanel>
      </Expander>
      <Button Width="80">
         Ok
      </Button>
   </StackPanel>
</Page>
```



◄ **Figure 4-28 :** Utilisation d'Expander

Pour ouvrir ou fermer un `Expander`, il suffit d'utiliser les flèches sur sa droite ; l'orientation des flèches indique s'il est ouvert ou fermé. Si les flèches sont dirigées vers le bas, le contenu est caché. *A contrario*, si les flèches sont dirigées vers le haut, le contenu est affiché.



◄ **Figure 4-29 :** L'utilisateur a ouvert tous les Expander

Comme vous pouvez le constater, il est possible de mettre plusieurs `Expander` dans une même page. En revanche, un `Expander` ne peut avoir qu'un seul nœud enfant. Celui-ci sera dès lors souvent un conteneur comme un `StackPanel`. Notez dans l'exemple l'utilisation combinée de `StackPanel` et de `WrapPanel`.

L'attribut `Header` permet de déterminer le titre, alors que l'attribut `IsExpanded` indique si la zone est cachée (`false`) ou affichée (`true`). Par défaut, le contenu est caché.

Notez que le bouton ne sera jamais caché car il n'est pas repris dans une balise `Expander`.

Astuce

Direction d'expansion

Si vous souhaitez que la zone s'ouvre vers le haut plutôt que vers le bas, utilisez l'attribut `ExpandDirection` et assignez-lui la valeur `Up`.

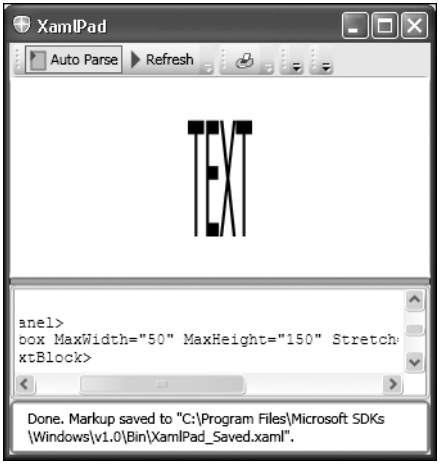
4.10 Utiliser une ViewBox

La ViewBox permet de définir une zone écran dont le contenu pourra être automatiquement adapté à la taille de la zone. L'adaptation se fait en étendant verticalement, horizontalement ou dans les deux sens le contenu.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel>
      <Viewbox MaxWidth="150" MaxHeight="150"
       Stretch="Fill" StretchDirection="Both">
         <TextBlock>
            TEXT
         </TextBlock>
      </Viewbox>
   </StackPanel>
</Page>
```

◄ **Figure 4-30 :** Affichage d'une ViewBox

Grâce à l'option Stretch="Fill", le contenu occupe automatiquement l'entièreté de la ViewBox. L'attribut StretchDirection vous permet de limiter l'agrandissement au sens horizontal ou vertical. Si vous changez la largeur de 150 en 50, le contenu est modifié en fonction.

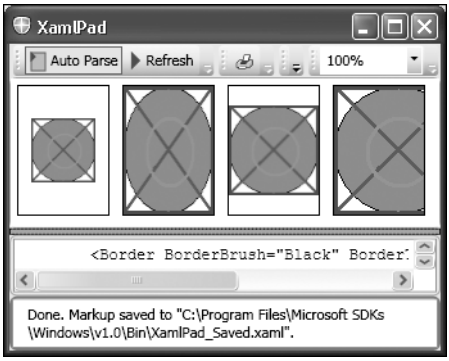


◄ **Figure 4-31** : La même ViewBox modifiée

Selon la valeur donnée à l'attribut `Stretch`, le résultat sera fort différent. Pour bien comprendre le fonctionnement de chacune de ces valeurs, voici un autre exemple.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <WrapPanel>
      <Border BorderBrush="Black" BorderThickness="1"
       Margin="5,5,5,5">
         <Viewbox Width="70" Height="100"
          Stretch="None" StretchDirection="Both">
            <Image Source="c:\Mir.bmp" Width="50"
             Height="50"/>
         </Viewbox>
      </Border>
      <Border BorderBrush="Black" BorderThickness="1"
       Margin="5,5,5,5">
         <Viewbox Width="70" Height="100"
          Stretch="Fill" StretchDirection="Both">
            <Image Source="c:\Mir.bmp" Width="50"
             Height="50"/>
         </Viewbox>
      </Border>
      <Border BorderBrush="Black" BorderThickness="1"
       Margin="5,5,5,5">
         <Viewbox Width="70" Height="100"
          Stretch="Uniform" StretchDirection="Both">
```

```
            <Image Source="c:\Mir.bmp" Width="50"
             Height="50"/>
          </Viewbox>
        </Border>
        <Border BorderBrush="Black" BorderThickness="1"
         Margin="5,5,5,5">
          <Viewbox Width="70" Height="100"
           Stretch="UniformToFill" StretchDirection="Both">
            <Image Source="c:\Mir.bmp" Width="50"
             Height="50"/>
          </Viewbox>
        </Border>
    </WrapPanel>
</Page>
```



◄ **Figure 4-32** : Effet de l'attribut Stretch

Pour une question de clarté de l'exemple, la taille de l'image est fixée dans le code à 50 sur 50 et est placée dans quatre `ViewBox` identiques à l'exception de la valeur de l'attribut `Stretch`. Les balises `Border` sont uniquement présentes pour délimiter visuellement l'emplacement des `ViewBox`. Le résultat parle de lui-même.

4.11 Utiliser un Popup

Le Popup permet d'ouvrir une zone écran contenant n'importe quel contenu XAML. Il est parfois confondu à tort avec les tooltips, dont il peut simuler l'utilisation. En effet, l'attribut `PlacementTarget` permet d'associer le Popup à un autre élément de l'écran tandis que `PlacementRectangle` permet d'en définir les dimensions. Pour utiliser un Popup, il faut aussi utiliser du code .NET.

Pour bien comprendre son utilisation, il est nécessaire de comprendre également la gestion des événements. Si tel n'est pas votre cas, sachez seulement que, lorsque vous cliquez sur le bouton **Cliquer pour ouvrir**, la méthode

AffichePopup est exécutée et que, lorsque vous cliquez sur le bouton **Cliquer pour fermer**, c'est la méthode CachePopup qui est exécutée.

Renvoi *La gestion des événements sera abordée dans un chapitre spécifique page 138.*

```
<Page x:Class="Page1"
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel>
      <Button Name="MonBouton" Click="AffichePopup">
         Cliquez pour ouvrir
      </Button>
      <Popup Name="MonPopup"
       PlacementTarget="{Binding ElementName=MonBouton}"
       PlacementRectangle="30,20,100,40">
         <Border BorderThickness="1" BorderBrush="Black">
            <StackPanel Margin="5,5,5,5"
             Background="AliceBlue">
               <Label>
                  Ceci n'est pas un tooltip mais
                  un popup et peut être une page
                  à part entière
               </Label>
               <Button Click="CachePopup">
                  Cliquez pour fermer
               </Button>
            </StackPanel>
         </Border>
      </Popup>
   </StackPanel>
</Page>


' Interaction logic for Page1.xaml
Partial Public Class Page1
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub AffichePopup(ByVal sender As Object, ByVal e As
    ➥ RoutedEventArgs)
        Me.MonPopup.IsOpen = True
    End Sub
```

```
    Public Sub CachePopup(ByVal sender As Object, ByVal e As
    ➥ RoutedEventArgs)
        Me.MonPopup.IsOpen = False
    End Sub
End Class
```

◄ **Figure 4-33** : Le Popup fermé

Comme vous pouvez le constater, rien ne suggère qu'il puisse y avoir un Popup.

◄ **Figure 4-34** : Le Popup ouvert

4.12 Ajouter de la vidéo dans la fenêtre

Pour visionner une vidéo ou simplement jouer de la musique, vous pouvez utiliser la classe `MediaElement`. Elle s'utilise assez simplement mais dispose bien sûr de propriétés et méthodes pour la gestion telles simplement les méthodes `Play`, `Pause` et `Stop`.

```
<Page xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <Grid Background="Black">
      <Border Width="300" Height="200" Name="Movie"
       BorderBrush="White" CornerRadius="2"
       BorderThickness="2">
         <MediaElement LoadedBehavior="Play"
          Source="c:\News.wmv" />
      </Border>
   </Grid>
</Page>
```

◄ **Figure 4-35 :** Visionner une vidéo

`MediaElement` est simplement placée dans un cadre. La propriété `Source` permet de définir le fichier multimédia qui devra être utilisé. La propriété `LoadedBehavior` permet de déterminer l'action qui sera prise directement après le chargement du contrôle. La propriété `UnloadedBehavior` a le même effet mais lors du déchargement du contrôle. Les valeurs possibles sont `Stop`, `Play`, `Close`, `Pause`, `NoSet` et `Manual`.

Écran vide

Il est possible que l'emplacement où devrait se trouver la vidéo reste vide. Il s'agit d'un problème de pilote de la carte écran qui n'est pas adapté. En effet, `MediaElement` requiert un pilote récent et adapté. Si c'est votre cas, essayez de mettre à jour votre pilote. C'est aussi pour cette raison que l'écran vous est présenté sous Windows Vista et non sous Windows XP comme les autres écrans.

Il existe également une classe `SoundPlayer` et une `MediaPlayer`. `SoundPlayer` ne peut être utilisée directement en XAML mais uniquement en code .NET. De manière générale, l'utilisation de `MediaElement` est très largement privilégiée, et tout particulièrement dans le code XAML.

Il est possible également d'utiliser les `Triggers` pour manipuler la vidéo depuis, par exemple, des boutons. Dans l'exemple qui suit, un bouton `Play` permet de démarrer la vidéo et un bouton `Pause` permet de l'arrêter.

Si vous voulez plus d'informations sur les triggers, reportez-vous au chapitre ***Utiliser les Triggers*** *page 238.*

```
<Window x:Class="Window1" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="WindowsApplication8" Height="300" Width="300"
 >
   <Window.Triggers>
```

Le trigger suivant sera déclenché lors du clic sur le bouton `btnPlay`.

```
      <EventTrigger RoutedEvent="Button.Click"
       SourceName="btnPlay">
         <EventTrigger.Actions>
            <BeginStoryboard Name= "Debut">
               <Storyboard>
```

Le storyboard a défini une `MediaTimeLine` qui est une `TimeLine` spécifique aux éléments multimédias. C'est ici qu'est définie la source.

```
                  <MediaTimeline Source="c:\news.wmv"
                   Storyboard.TargetName="laVideo"/>
               </Storyboard>
            </BeginStoryboard>
         </EventTrigger.Actions>
      </EventTrigger>
```

Le trigger suivant sera déclenché lors du clic sur le bouton btnPause.

```
      <EventTrigger RoutedEvent="Button.Click"
       SourceName="btnPause">
         <EventTrigger.Actions>
```

La commande suivante met en pause le storyboard appelé Debut.

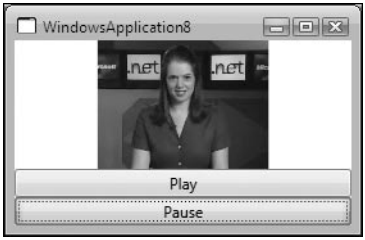
```
            <PauseStoryboard
             BeginStoryboardName="Debut" />
         </EventTrigger.Actions>
      </EventTrigger>
   </Window.Triggers>
   <StackPanel>
```

Aucune source n'est définie pour le MediaElement.

```
      <MediaElement Height="100" Width="150" Name="laVideo"/>
      <Button Name="btnPlay" >Play</Button>
      <Button Name="btnPause" >Pause</Button>
   </StackPanel>
</Window>
```

◄ **Figure 4-36 :** Fenêtre de visualisation

Si vous cliquez sur le bouton **Play**, la vidéo démarre.



◄ **Figure 4-37 :** La vidéo affichée

Remarque

Méthodes de `MediaElement`

Vous pouvez remarquer qu'à aucun moment nous n'avons utilisé les méthodes `Start` et `Pause` de la classe `MediaElement`.

4.13 Checklist

Ce chapitre a été l'occasion de passer en revue les différents contrôles permettant de réaliser une interface utilisateur complète. C'est-à-dire :

- créer et manipuler les listes déroulantes, que ce soient les `ListBox` ou les `ComboBox` ;
- utiliser les cases à cocher (`CheckBox`) ;
- utiliser les boutons radio (`RadioButton`) et les grouper ;
- créer des info-bulles (`ToolTip`) ;
- créer un panneau à onglets (`TabControl`) ;
- configurer un bouton pour le rendre autorépétitif ;
- utiliser un curseur (`Slider`) pour déterminer une valeur ;
- utiliser des `Expander` pour rendre l'interface plus lisible ;
- utiliser une `ViewBox` ;
- créer des fenêtres `Popup` ;
- manipuler le multimédia avec `MediaElement`.

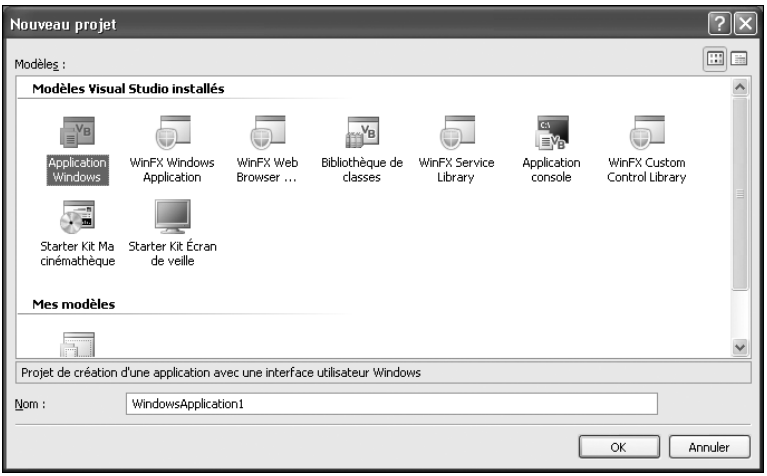
Créer une application

Créer une application Windows 132
Gérer les événements 138
Héberger une application dans un browser 140
Les pages fonctions 149
Créer une application Windows navigable 157
Les applications avec WPF/E 165
Checklist 167

5.1 Créer une application Windows

Après avoir manipulé le XAML avec XAMLPad, voyons maintenant comment réaliser une application. Dans le domaine, nous jouerons la conformité avec le très traditionnel "Hello World".

Pour commencer, ouvrez Visual Studio. Choisissez **Fichier**, **Nouveau**, **Projet**.



▲ **Figure 5-1 :** Choix du type de projet

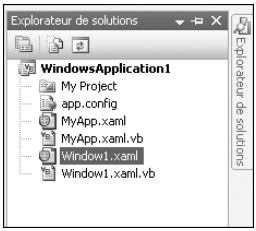
Dans la fenêtre des projets, choisissez le type *WinFX Windows Application*.

Remarque

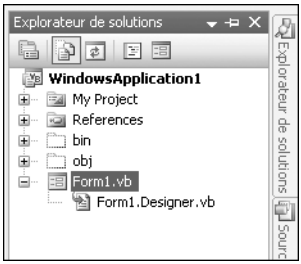
Ce type n'est pas présent

Si vous ne trouvez pas ce type de projet dans Visual Studio, c'est que vous n'avez pas installé le kit de développement ou, selon les versions, l'*add-in* pour Visual Studio. Vous le trouverez sur le site de Microsoft.

Donnez un nom à l'application, par exemple `HelloWorld`.



◄ **Figure 5-2 :** L'explorateur de solution dans un projet WinFX



◄ **Figure 5-3 :** L'explorateur de solution dans un projet WinForm

Au lieu du traditionnel *Form1.vb*, vous retrouvez quatre fichiers différents. Ils sont associés par paire : les fichiers *MyApp.xaml* et *MyApp.xaml.vb*, d'une part, et les fichiers *Window1.xaml* et *Window1.xaml.vb*, d'autre part. Chaque paire de fichiers prend en charge la gestion d'une classe. Il est vrai que, si vous avez déjà travaillé avec Visual Studio 2005, vous savez que derrière le fichier *Form1.vb* se cachait un fichier *Form1.Designer.vb*. Le but dans les deux cas est le même : séparer le code d'affichage du code de traitement.

Mais revenons à notre projet et examinons pour commencer le fichier *MyApp.xaml*.

```
<Application x:Class="MyApp"
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="Window1.xaml">
   <Application.Resources>

   </Application.Resources>
</Application>
```

A priori, il est probable que vous n'ayez pas à changer ce fichier sauf l'attribut `StartupUri`, qui définit la classe à instancier et à exécuter au démarrage de l'application. Il ne serait pas de bon ton de conserver pour votre classe le nom par défaut. Remplacez *Window1.xaml* par *HelloWorld.xaml*.

Le fichier code .Net associé est encore plus simple.

```
' Interaction logic for MyApp.xaml
Partial Public Class MyApp
    Inherits Application

End Class
```

Comme vous le voyez, à ce niveau rien à dire. Dans la majorité des cas, vous n'aurez pas à ajouter du code dans cette partie. N'hésitez pas, toutefois, à

changer le nom `MyApp` en un nom plus cohérent. Faites-le également dans le fichier xaml et renommez aussi les fichiers.

Attention

Propriétés de l'application

Quand vous changez le nom du fichier *MyApp*, la propriété **Action de génération** du fichier devient `Page`. Vous devez remettre la valeur `ApplicationDefinition`. Pour y accéder, cliquez du bouton droit sur le nom du fichier et choisissez **Propriétés**.

◄ **Figure 5-4 :** Propriétés d'un fichier

Astuce

Nom des classes

Si le nom des classes diffère entre le fichier XAML et le fichier *xaml.vb*, vous aurez immanquablement une erreur à la compilation. Les fichiers peuvent porter un autre nom que la classe mais il est de bon aloi de garder le même.

Passons maintenant au fichier décrivant la fenêtre de l'application.

Remarque

Nom des classes et des fichiers

Les classes et les fichiers ont été renommés pour porter le nom `HelloWorld`.

```
<Window x:Class="HelloWorld"
    xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Ma première application"
    >
    <Grid>

    </Grid>
</Window>
```

Comme vous pouvez le constater, Microsoft vous propose par défaut l'utilisation d'une grille. Contrairement aux exemples vus jusqu'à présent, la première balise est une balise `Window` et non plus une balise `Page`. La balise `Window` est utilisée pour définir une fenêtre de type Windows. Nous reviendrons ultérieurement sur l'utilisation de la balise `Page`.

Ajoutons le code suivant entre les balises `Grid`.

```
<Label VerticalAlignment="Center"
 HorizontalAlignment="Center">
   Bonjour le monde.
</Label>
```

Nous pouvons maintenant exécuter l'application. Appuyez sur la touche F5.

◀ **Figure 5-5 :** Ma première application

La balise `Window` dispose comme les autres balises d'un grand nombre d'attributs. Certains comme `Width` ou `MinHeight` ont déjà été vus dans d'autres chapitres et nous ne reviendrons pas dessus. En revanche, il existe quelques autres attributs fort intéressants.

Vous avez déjà pu voir dans les exemples l'action de l'attribut `Title`, qui permet de définir le titre de la fenêtre.

Pour positionner la fenêtre, vous disposez de l'attribut `WindowsStartupLocation`, qui va déterminer la position initiale de la fenêtre.

```
WindowStartupLocation="CenterScreen"
```

Les valeurs possibles sont `CenterScreen` pour centrer la fenêtre dans l'écran, idéal pour la fenêtre principale ; `CenterOwner` pour centrer la fenêtre dans la fenêtre qui la contient, idéal pour les sous-fenêtres ; et `Manual` pour vous permettre de choisir la position vous-même. Cette dernière valeur est à utiliser en conjonction avec les attributs `Top` et `Left`.

```
<Window x:Class="HelloWorld"
    xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Ma première application"
    WindowStartupLocation="Manual"
    Top="100"
    Left="300"
    >
```

Un autre attribut intéressant est l'attribut `SizeToContent`. S'il est fixé à `True`, la taille de la fenêtre lors de la création sera automatiquement adaptée au contenu.

```
SizeToContent="WidthAndHeight"
```

◄ **Figure 5-6 :** Utilisation de SizeToContent

Il est possible de n'adapter la taille qu'en largeur en utilisant la valeur `Width` ou uniquement en hauteur en utilisant la valeur `Height`.

Vous pouvez également opter pour démarrer automatiquement en mode maximisé ou minimisé.

```
WindowState="Maximized"
```

Il est également possible de modifier le comportement et les possibilités de redimensionnement des fenêtres en utilisant l'attribut `ResizeMode`.

```
ResizeMode="CanResizeWithGrip"
```

◄ **Figure 5-7 :** Utilisation de ResizeMode

Notez le grip en bas à droite.

Vous pouvez également opter pour interdire le redimensionnement avec la valeur `NoResize`.

La taille de la fenêtre n'est pas la seule chose que nous puissions modifier, il est également possible de modifier son affichage en ajoutant l'attribut `WindowStyle`. Par défaut, sa valeur est SingleBorderWindow, mais vous pouvez ajouter un effet 3D.

```
WindowStyle="ThreeDBorderWindow"
```

◄ **Figure 5-8 :**
Fenêtre avec effet 3D

Ou lui donner le look d'une fenêtre d'outils.

```
WindowStyle="ToolWindow"
```

◄ **Figure 5-9 :**
Fenêtre d'outils

Ou encore afficher la fenêtre sans aucun bord.

```
WindowStyle="None"
```

◄ **Figure 5-10 :**
Fenêtre sans bordure

Pour certaines fenêtres, vous souhaiterez peut-être qu'il ne soit pas possible de les recouvrir avec une autre. Si c'est le cas, utilisez l'attribut `Topmost`.

```
Topmost="True"
```

Par défaut, toutes les fenêtres sont automatiquement reprises dans la barre des tâches. Si, pour la fenêtre principale, cette option par défaut est judicieuse, ce n'est pas toujours le cas pour les autres fenêtres. Pour éviter qu'une fenêtre ne soit reprise dans la barre des tâches, il suffit d'utiliser l'attribut `ShowInTaskbar`.

```
ShowInTaskbar="False"
```

5.2 Gérer les événements

Pour illustrer l'utilisation des événements, reprenons l'exemple précédent. Pour une simple question de facilité, dans ce nouvel exemple la grille est toutefois remplacée par un `StackPanel`.

```
<Window x:Class="HelloWorld"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Ma première application"
 WindowStartupLocation="Manual"
 Top="100"
 Left="300"
 Name="HelloWorld"
 >
   <StackPanel HorizontalAlignment="Center"
    VerticalAlignment="Center">
     <Label Name="lblTexte">
        Bonjour le monde.
     </Label>
     <Button Name="btnRepondre">
        Répondre
     </Button>
   </StackPanel>
</Window>
```

Attention

Centrage

Les attributs réalisant le centrage sont déplacés du `Label` au `StackPanel`. Sans cela, vu le type de fonctionnement du `StackPanel`, le centrage vertical dans la fenêtre n'aurait pas lieu.

Un bouton a également été ajouté à la fenêtre.

La question est maintenant de savoir comment associer une action au bouton. Comme vous pouvez l'imaginer, l'action sera réalisée dans le code .NET au sein d'une méthode de la classe. Ceci étant entendu, le problème se résume alors à associer le bouton à la méthode voulue. Tout d'abord, il faut savoir que la méthode doit être associée non pas au bouton lui-même mais à un événement du bouton. En l'occurrence, l'événement `Click`.

Commençons par écrire dans le code VB une méthode que vous désirez exécuter.

```
Public Sub Click_Repondre(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.lblTexte.Content = "Merci"
        Me.btnRepondre.IsEnabled = False
End Sub
```

Cette méthode va remplacer le texte `"Bonjour le monde"` par `"Merci"`.

La signature de la méthode, c'est-à-dire les différents paramètres, doit absolument respecter la définition reprise dans l'exemple. Elle correspond par ailleurs strictement à la définition des méthodes chargées de répondre aux événements dans l'environnement .NET. Le nom est quant à lui entièrement libre, mais choisissez toujours un nom parlant.

Astuce

Utiliser l'IntelliSense

Pour bénéficier de l'IntelliSense et ainsi retrouver facilement les noms des membres que vous aurez définis dans votre code XAML, n'hésitez pas à utiliser le mot clé ME même s'il est facultatif.

Maintenant que nous avons défini la méthode, il ne reste plus qu'une simple étape à faire, réaliser l'association entre l'événement et la méthode. Nous pourrions classiquement le faire en .NET mais il est plus judicieux de le faire dans le code XAML.

```
<Button Name="btnSuite" Click="Click_Repondre">
   Suite
</Button>
```

C'est aussi simple que cela. Le nom de l'attribut correspond au nom de l'événement et la valeur est le nom de la méthode à appeler.

◄ **Figure 5-11 :** Cliquer sur Répondre

◄ **Figure 5-12 :** Répondre a été cliqué

La même technique peut être appliquée avec n'importe quel événement de n'importe quelle classe.

5.3 Héberger une application dans un browser

Aperçu de cette technologie

Il est possible de créer avec XAML des applications qui s'exécutent non pas dans une fenêtre Windows mais dans un navigateur Internet. On parle alors de smart client. Ce type d'application est également appelé *Web Browser Application* ou WBA. Il s'agit d'une application online qui s'exécute dans un browser et qui n'est pas installée localement. En résumé, en accédant à une URL le programme est téléchargé et exécuté sans qu'il ne soit installé. Contrairement à une application ASP.NET, le programme s'exécute sur le poste client et non sur le serveur. Il fait le lien avec les serveurs de données exactement comme le ferait une application Windows classique.

Comme l'application s'exécute sur le poste client, elle n'est plus soumise aux restrictions imposées par le HTML. Vous pouvez dans ce type d'application

tirer parti de toute la puissance des API WinFX ou presque, et ce y compris les fonctionnalités 3D. Toutefois, comme pour l'ASP.NET, le développement se fait en mode Page. Contrairement à une application ASP.NET, les pages sont codées dans un même module. Leur accès se fera alors sans devoir faire appel au serveur.

En revanche, le fait que l'application soit exécutée sur le poste client rend nécessaire la présence de WinFX sur le poste client. Il ne s'agit donc pas d'une technologie universelle indépendante de la plate-forme utilisée comme l'est le HTML.

La sécurité et les WBA

L'application n'est pas installée sur votre PC mais est exécutée quand vous accédez à une adresse web depuis votre navigateur. Donc, *a priori*, vous ne connaissez pas forcément l'application qui va s'exécuter. Ce qui pourrait entraîner de graves risques pour la sécurité de votre PC.

Pour ces raisons, l'application est exécutée dans une *Sandbox* (aussi appelé *bac à sable*) ; c'est-à-dire un environnement où les privilèges attribués à l'application sont réduits. Par défaut, l'application sera exécutée dans la zone Internet. De cette manière, la sécurité de votre ordinateur est assurée au même titre qu'avec une application web traditionnelle.

Le fait d'exécuter l'application dans un environnement fermé a pour conséquence que certaines fonctionnalités ne peuvent être utilisées. Les droits attribués à la *Sandbox* peuvent être étendus en modifiant la zone d'exécution, comme vous pouvez le faire avec n'importe quel site. Par exemple, vous pouvez placer l'application dans la zone intranet au lieu de la zone Internet. De cette manière, vous allez récupérer un certain nombre de fonctionnalités.

Héberger et exécuter ce type d'application

L'application est stockée sur un serveur web. En ce qui concerne IIS, vous devez disposer de la version 5 ou supérieure. Le serveur ne doit pas nécessairement contenir WinFX. En revanche, un minimum de configuration est nécessaire. La configuration du serveur dépasse largement le cadre de cet ouvrage mais, si vous êtes intéressé, vous pouvez trouver les informations nécessaires sur le site de Microsoft.

Sur le client, WinFX doit être installé. Le browser doit être Internet Explorer 6 ou supérieur ou n'importe quel browser qui implémente Microsoft WebBrowser Control.

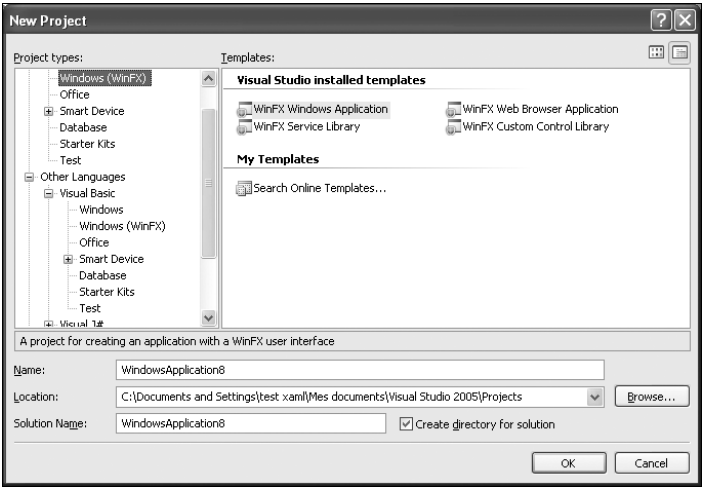
Vous pouvez également parfaitement exécuter des applications de ce type et qui sont présentes sur votre PC.

Quand recourir à ce modèle d'application ?

Par ses spécificités, les WBA sont particulièrement utiles pour les applications à contenu ou logique complexe pour lesquelles vous ne souhaitez pas installer un client local. Là où les deux clients doivent exister, le fait que le code puisse être facilement transposé entre smart client et client riche est aussi un atout. Il est clair qu'il sera plus aisé d'utiliser ce type d'application dans un environnement intranet homogène.

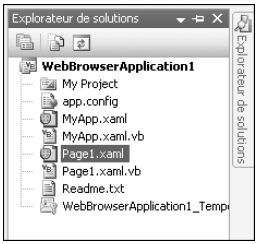
Créer une WBA

Pour créer une WBA dans Visual Studio, vous devez créer un nouveau projet et choisir le type WinFX Web Browser Application, anciennement connue sous le nom d'Express Application.



▲ **Figure 5-13 :** Créer une WBA

Le contenu du projet est très semblable au contenu pour une application Windows.



◄ **Figure 5-14 :** Contenu d'une WBA

Vous y retrouvez les deux paires de fichiers mais, cette fois, au lieu de *Window1* les fichiers se nomment *Page1*.

Voyons le contenu de ces quatre fichiers et les différences par rapport à ceux réalisés dans l'application Windows (voir p. 132).

Si ce n'est l'adresse de l'URI, les fichiers *MyApp.xaml* et *MyApp.xaml.vb* sont identiques à ceux déjà vus.

```
<Application x:Class="MyApp"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 StartupUri="Page1.xaml"
 >
   <Application.Resources>

   </Application.Resources>
</Application>
```

```
' Interaction logic for MyApp.xaml
Partial Public Class MyApp
    Inherits Application


End Class
```

En revanche, les fichiers *Page1.xaml* et *Page1.xaml.vb* héritent non plus d'un objet de classe `Window` mais bien d'un objet de la classe `Page`.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <Grid>

   </Grid>
</Page>
```

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
```

```
Imports System.Windows.Shapes

' Interaction logic for Page1.xaml
Partial Public Class Page1
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

End Class
```

Il n'y a en définitive pas de différences notables.

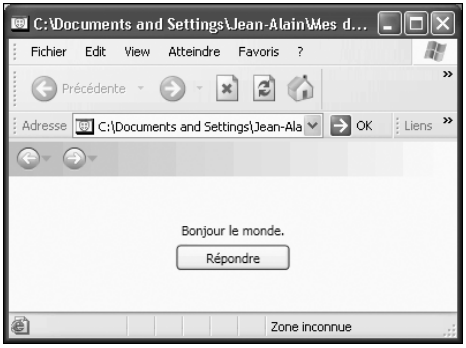
Pour changer les noms des classes et des fichiers, vous devez utiliser la technique décrite dans le paragraphe ***Créer une application Windows*** *page 132.*

Maintenant, intégrons le code utilisé dans notre application Windows.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <StackPanel HorizontalAlignment="Center"
    VerticalAlignment="Center">
     <Label Name="lblTexte">
        Bonjour le monde.
     </Label>
     <Button Name="btnRepondre" Click="Click_Repondre">
        Répondre
     </Button>
   </StackPanel>
</Page>

Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes

' Interaction logic for Page1.xaml
Partial Public Class Page1
```

```
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Click_Repondre(ByVal sender As Object _
                        , ByVal e As RoutedEventArgs)
        Me.lblTexte.Content = "Merci"
        Me.btnRepondre.IsEnabled = False
    End Sub

End Class
```

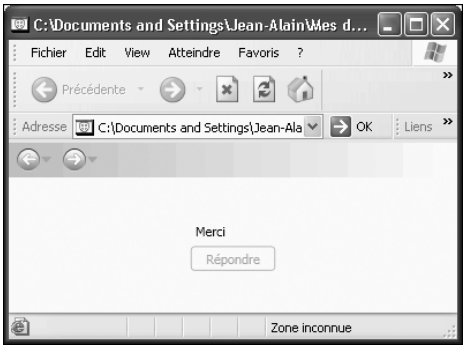
Exécutez l'application en utilisant par exemple la touche F5.



◄ **Figure 5-15 :** Exécution d'une WBA

L'application est très semblable à l'application Windows mais elle est bien exécutée dans le navigateur.

Si nous cliquons sur le bouton **Répondre**, l'application réagit comme prévu.



◄ **Figure 5-16 :** Événement activé dans la WBA

Astuce

Erreur de sécurité

Si vous utilisez une version express de Visual Studio, vous recevrez peut-être une erreur de sécurité lors de la première exécution. Fermez puis rouvrez votre projet et l'erreur disparaîtra d'elle-même.

Si dans votre page vous ne devez pas utiliser de méthode, vous pouvez opter pour une page XAML seule. Il s'agit alors d'une page statique. Le code XAML sera directement exécuté lors du chargement de la page.

Faites un copier-coller du code XAML dans **NotePad**. Supprimez l'attribut `Class` et l'attribut `Click` afin d'obtenir le code suivant :

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
    >
   <StackPanel HorizontalAlignment="Center"
     VerticalAlignment="Center">
      <Label Name="lblTexte">
         Bonjour le monde.
      </Label>
      <Button Name="btnRepondre">
         Répondre
      </Button>
   </StackPanel>
</Page>
```

Sauvez le document et, depuis l'Explorateur Windows, double-cliquez dessus ou tapez son adresse dans Internet Explorer. La même page est chargée mais, bien entendu, le bouton n'effectue plus aucune action.

Comme il est possible de modifier la taille d'une fenêtre Windows, il est également possible de modifier la taille de la page mais aussi de la fenêtre du navigateur. Les attributs `WindowWidth` et `WindowHeight` permettent de contrôler la taille de la fenêtre du navigateur alors que les attributs `Width` et `Height` contrôlent la taille de la page. Modifiez les valeurs de ces attributs dans l'exemple ci-dessous pour bien comprendre leur portée.

```
<Page x:Class="Page1"
    xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page1"
    WindowTitle="Ma page"
```

```
    WindowWidth="500"
    WindowHeight="300"
    Width="470"
    Height="200"
 >
   <Border BorderThickness="1" BorderBrush="Black">
      <StackPanel HorizontalAlignment="Center"
       VerticalAlignment="Center">
        <Label Name="lblTexte">
           Bonjour le monde.
        </Label>
        <Button Name="btnRepondre" Click="Click_Repondre">
           Répondre
        </Button>
      </StackPanel>
   </Border>
</Page>
```

Remarque

Titre de la fenêtre

Le titre de la fenêtre est donné non pas par l'attribut `Title` de la balise `Page` mais bien par l'attribut `WindowTitle`.

Enchaînement des pages

Pour naviguer entre les pages, WinFX met à notre disposition la classe `NavigationService`. Afin d'illustrer son utilisation, nous allons remplacer dans l'exemple précédent l'affichage du mot "merci" dans l'étiquette par l'affichage d'une page "Merci".

Créons tout d'abord la nouvelle page à afficher.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page2"
 >
   <StackPanel HorizontalAlignment="Center"
    VerticalAlignment="Center">
     <Label Name="lblTexte">
        Merci.
     </Label>
   </StackPanel>
</Page>
```

La page étant entièrement statique, il est inutile de créer une contre-partie VB. Évidemment, si votre seconde page est plus élaborée, rien n'empêche de créer une page complète avec du contenu XAML et .NET.

Maintenant, voyons comment réaliser l'appel de cette page depuis le code .NET de notre première page.

```
Public Sub Click_Repondre(ByVal sender As Object _
                        , ByVal e As RoutedEventArgs)
   Me.NavigationService.Navigate(New Uri("Page2.xaml", UriKind.Relative))
End Sub
```

L'appel à la page se fait comme précédemment dans l'événement `Click` associé au bouton en utilisant la méthode `Navigate` de la propriété `NavigationService`. Cette méthode reçoit comme paramètre une URI qui donne l'adresse de la page à afficher. L'URI peut être absolue ou relative.

◄ **Figure 5-17 :** Navigation entre pages

Attention

Barre de navigation d'Internet Explorer

Notez que la barre de navigation du navigateur ne permet pas de réaliser les habituelles opérations **Précédente, Suivante**. Vous devez pour cela utiliser la barre de navigation spécifique.

◄ **Figure 5-18 :** Barre de navigation

Si vous ne souhaitez pas voir apparaître la barre de navigation, il suffit d'assigner `False` à l'attribut `ShowsNavigationUI` de la balise `Page`.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 ShowsNavigationUI="False"
>
```

◄ **Figure 5-19** : Sans barre de navigation

`NavigationService` dispose également des méthodes `GoBack` et `GoForward` pour provoquer un passage à la page précédente ou à la page suivante. Les méthodes `CanGoBack` et `CanGoForward` vous informent sur la possibilité ou non d'appeler les méthodes respectives.

5.4 Les pages fonctions

Les pages fonctions sont un type de page particulier. Ce type de page est appelé depuis une autre page et peut recevoir des paramètres. Il dispose d'une méthode `OnReturn` qui provoque le retour dans la page appelante et qui, de plus, permet de renvoyer une valeur. Il n'est pas nécessaire de connaître la page appelante. Pour récupérer la valeur en retour, il faudra utiliser la gestion des événements.

Nous aurons besoin de cette petite classe utilitaire développée pour l'exemple ci-après.

```
Public Class Data
   Public Val As Integer
   Public Texte As String
End Class
```

Il n'y a pas grand-chose à dire sur cette classe. Elle servira pour le transfert de données entre les pages.

La première page qui est automatiquement chargée lors du démarrage de l'application est une page classique qui hérite de `Page`.

```
<Page x:Class="Page1" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <StackPanel>
      <Label Name="valeur"/>
      <Label Name="texte"/>
      <Button Name="btnPageSuiv" Click="PageSuivante">
         Page suivante
      </Button>
   </StackPanel>
</Page>
```

Dans le fichier *Page1.xaml*, deux Label sont définies mais sont vierges. Elles serviront à afficher les valeurs reçues. Le bouton sert à demander la navigation vers la page suivante. L'appel se fera dans la méthode `PageSuivante`.

```
Partial Public Class Page1
    Inherits Page
```

Dans le code .NET associé, nous déclarons un membre du type de la page qui sera appelé. Il est important de déclarer que la page se mettra à l'écoute des événements, ce qui est réalisé avec `WithEvents`.

```
    Private WithEvents pageSuiv As Page2

    Public Sub New()
        InitializeComponent()
    End Sub

    Sub PageSuivante(ByVal sender As Object, ByVal e As RoutedEventArgs)
```

La transmission des paramètres à la page se fait par le constructeur. L'appel de la page elle-même se fait exactement comme une page normale.

```
        pageSuiv = New Page2("défaut")
        Me.NavigationService.Navigate(pageSuiv)
    End Sub
```

Il faut définir la méthode qui sera appelée lors du retour. Pour qu'elle soit effectivement appelée, il faut l'associer à l'événement avec `Handles`. Notez que

le type de la valeur reçue est précisé.

```
    Private Sub return_handler( _
           ByVal sender As Object, _
           ByVal e As ReturnEventArgs(Of Data)) _
           Handles pageSuiv.Return
```

Les valeurs reçues sont placées dans les contrôles `Label` et le bouton est désactivé.

```
        valeur.Content = e.Result.Texte
        texte.Content = e.Result.Val
        btnPageSuiv.IsEnabled = False
    End Sub
End Class
```

Nous devons maintenant définir la page 2, qui sera une `PageFunction`. Notez la définition du type de paramètre et la présence de la déclaration du namespace de l'application. Le namespace étant dans l'assembly du programme, il n'est pas nécessaire de préciser l'assembly. La `TextBox` recevra la valeur reçue en paramètre, qui pourra ainsi être modifiée. La méthode associée à l'événement `Click` du bouton aura pour effet de fermer la page.

```
<PageFunction x:Class="Page2" x:TypeArguments="app:Data"
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:app="clr-namespace:WinFxBrowserApplication1"
 Title="Page2"
 >
   <StackPanel>
      <TextBox Name="texte"/>
         <Button Click="Fin">
            Fin
         </Button>
   </StackPanel>
</PageFunction>
```

Même dans le code .NET, dés la déclaration de la page, il faut décrire les paramètres reçus.

```
Partial Public Class Page2
    Inherits PageFunction(Of Data)
    Private donnée As New Data
```

Le constructeur reçoit le paramètre.

```
    Public Sub New(ByVal initVal As String)
        InitializeComponent()
```

```
        donnée.Texte = initVal
        donnée.Val = 100
        texte.Text = initVal
    End Sub
```

Pour fermer la fenêtre et revenir à la page appelée, on utilise `OnReturn`, qui doit impérativement retourner un objet de type `ReturnEventArgs` qui est un type générique.

```
    Sub Fin(ByVal sender As Object, ByVal e As RoutedEventArgs)
        donnée.Texte = texte.Text
        Dim retour As New ReturnEventArgs(Of Data)(donnée)
        OnReturn((retour)
    End Sub

End Class
```

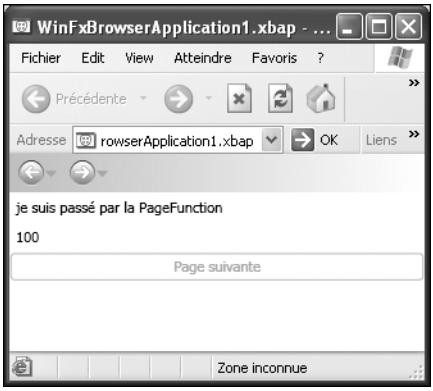
Lors du démarrage de l'application, nous recevons la première page.

◄ **Figure 5-20** : La page 1

Après avoir cliqué sur le bouton, la deuxième page est affichée. Et nous pouvons changer la valeur dans la boîte de texte.

◄ **Figure 5-21** : La page 2

Après avoir cliqué sur le bouton de fin, la première page est à nouveau affichée.



◄ **Figure 5-22 :** La page 1 modifiée

Il est également possible d'utiliser cette technique dans un `Frame`. Vous pouvez par exemple réaliser une fenêtre complète avec un menu et divers éléments qui doivent perdurer tout au long de la navigation et utiliser le `Frame` comme zone d'affichage des différents contenus. En somme, il s'agit d'un genre de page maître.

Pour illustrer cela, imaginons que notre page de démarrage est la page `StartPage` décrite ci-dessous.

```
<Page x:Class="StartPage" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="StartPage"
 >
   <DockPanel VerticalAlignment="Top">
      <Image DockPanel.Dock="Top"
         Source="c:\photo.jpg" MinWidth="300">
      </Image>
      <Menu DockPanel.Dock="Top" Height="30">
         <MenuItem Header="Page 1"
          Click="GoToPage1"/>
         <MenuItem Header="Page 2"
          Click="GoToPage2"/>
         <MenuItem Header="Page 3"
          Click="GoToPage3"/>
      </Menu>
      <StatusBar DockPanel.Dock="Bottom"
       Background="LightBlue">
         <Label>
            Micro Application
         </Label>
```

```
      </StatusBar>
      <Frame Name="zoneContenu"
       Source="HomePage.xaml" MinHeight="150"/>
   </DockPanel>
</Page>
```

Il s'agit d'une page classique dont le menu servira à naviguer et contenant un `Frame` qui reçoit *HomePage.xaml*, dont le contenu est en définitive la page d'accueil.

```
Partial Public Class StartPage
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Sub GotoPage1(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim nvPage As New Page1
        zoneContenu.Navigate(nvPage)
    End Sub

    Sub GotoPage2(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim nvPage As New Page2
        zoneContenu.Navigate(nvPage)
    End Sub

    Sub GotoPage3(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim nvPage As New Page3
        zoneContenu.Navigate(nvPage)
    End Sub
End Class
```

Si nous examinons le code .NET de la page, nous pouvons constater qu'à chaque action du menu une nouvelle page est affichée en provoquant la navigation dans le `Frame` uniquement. Notre page maître reste donc bel et bien toujours affichée.

Remarque

Écoute des événements

Contrairement à l'exemple précédent, le programme appelant ne se met pas à l'écoute de la valeur de retour car, dans cet exemple, elle ne nous intéresse pas. Rien n'empêche de la faire si tel est le besoin.

La page d'accueil *HomePage.xaml* est également une page tout à fait classique.

```
<Page x:Class="HomePage" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel>
      <Label>
         Ceci est la page principale
      </Label>
   </StackPanel>
</Page>
```

Examinons maintenant la classe Page1. Page2 et Page3 sont construites sur le même modèle.

```
<PageFunction x:Class="Page1" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:sys="clr-namespace:System;assembly=mscorlib"
 x:TypeArguments="sys:String"
 >
   <StackPanel>
      <Label>
         Ceci est la page 1
      </Label>
      <Button Click="Retour">
         Retour
      </Button>
   </StackPanel>
</PageFunction>
```

Page1 est une page de type PageFunction construite de la même manière

Remarque

Référence au namespace System

Notez la présence d'une référence au namespace System et à l'assembly mscorlib pour pouvoir définir le type String comme argument.

```
Partial Public Class Page1
    Inherits PageFunction(Of String)

    Public Sub New()
        InitializeComponent()
    End Sub

    Sub Retour(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim retour As New ReturnEventArgs(Of String)("De Page1")
```

```
        OnReturn(retour)
    End Sub
End Class
```

Le code .NET est également extrêmement simple. La seule chose à noter est la présence de `OnReturn`, dont nous avons vu précédemment le fonctionnement.

Lors du lancement du programme, la page d'accueil est affichée.

◄ **Figure 5-23 :** Page d'accueil

Si dans le menu vous choisissez *Menu1*, la page 1 est à son tour affichée mais dans le `Frame`, laissant tout le contexte inchangé.

◄ **Figure 5-24 :** Page 1

Si vous cliquez sur `Retour`, la page d'accueil est à nouveau affichée.

Attention

Particularité de navigation

Si au lieu de cliquer sur *Retour* vous demandez une autre page dans le menu, par exemple pour demander la page 3, celle-ci s'affiche sans problème mais, dans le cas où vous cliquez à ce moment sur *Retour*, c'est non pas la page d'accueil qui est affichée mais bien la page 1, page qui est en réalité à la base de l'appel de la `PageFunction`.

Si vous souhaitez que la page d'accueil soit systématiquement réaffichée, il vous suffit de vous mettre à l'écoute de la valeur de retour, comme nous l'avons fait dans le premier exemple ; et, dans la méthode qui traite le retour, vous pouvez imposer d'afficher la page d'accueil dans le `Frame`.

Cette façon de travailler apporte énormément de souplesse. Rien ne vous empêche par exemple de travailler avec plusieurs `Frame`. Finalement, le modèle en page peut facilement suivre un comportement semblable aux applications Windows classiques. N'oubliez toutefois pas que l'utilisateur peut également utiliser la barre de navigation.

5.5 Créer une application Windows navigable

Il existe un troisième modèle d'application qui est un hybride entre les deux technologies vues ci-dessus. Il s'agit d'une application Windows classique dans le sens où elle est lancée depuis le PC client, s'exécute directement dans une fenêtre Windows et n'est donc pas soumise aux contraintes sécuritaires propres aux WBA. En revanche, il s'agit d'un modèle applicatif par navigation entre des pages exactement comme les WBA.

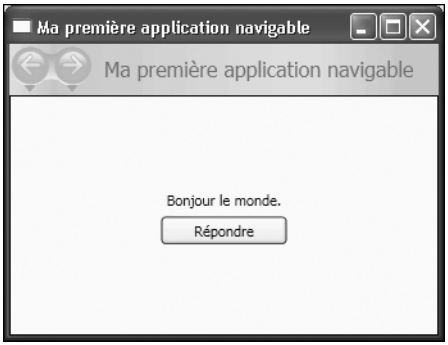
Ce modèle est parfait si vous désirez exécuter une application à la fois dans un browser et nativement dans Windows. Avec un minimum de modifications, une application WBA devient une application Windows navigable.

Voyons comment faire. Tout d'abord, vous devez créer une nouvelle application WinFX. Ensuite, importez dans ce projet les pages de votre application WBA.

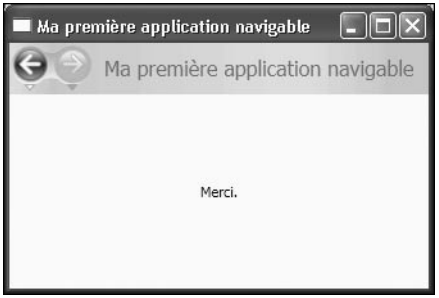
Si nous reprenons les pages réalisées dans l'exemple du chapitre précédent, nous devons donc avoir quatre fichiers XAML : *MyApp*, *Window1*, *Page1* et *Page2*.

Il ne nous reste maintenant plus qu'une seule chose à faire, remplacer le contenu de *Window1.xaml* par le code suivant :

```
<NavigationWindow x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Ma première application Windows en mode page"
 Source="Page1.xaml"
 />
```



◄ **Figure 5-25 :** Application navigable : première page



◄ **Figure 5-26 :** Application navigable : deuxième page

Attention

Window1.xaml.vb

Dans l'exemple, ce fichier est inutile et peut être effacé. Si vous désirez le conserver, vous devrez modifier l'héritage de la classe pour y faire apparaître `NavigationWindow` en lieu et place de `Window`.

L'utilisation d'une application WBA en tant qu'application Windows client n'est pas la seule utilité de `NavigationWindow`. Cette façon de travailler est aussi très utile pour réaliser un assistant Wizard, par exemple. Il s'agit non plus alors d'une application complète faite sur ce modèle mais uniquement d'une partie

de celle-ci. Le reste de l'application étant classiquement composé de fenêtres de type Window simple.

Voici un exemple extrêmement simplifié. Tout d'abord, la fenêtre classique, qui fait appel à l'assistant *via* un bouton, mais cela peut tout aussi bien être un menu.

```
<Window x:Class="ClassicWindow"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
      <Label Name="lblTexte">
         Fenêtre classique.
      </Label>
      <Button Name="btnWizard" Click="Wizard">
         Wizard
      </Button>
   </StackPanel>
</Window>

Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Shapes

' Interaction logic for Window1.xaml
Partial Public Class HelloWorld
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Wizard(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim wiz As New Window1
        wiz.ShowDialog()
    End Sub
End Class
```

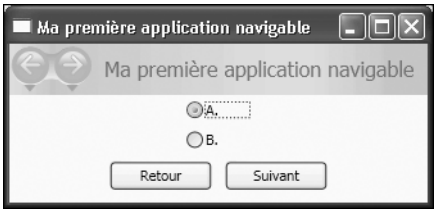
La méthode `Wizard` qui est associée à l'événement `Click` du bouton instancie et affiche une fenêtre de type `Window1`. Comme vous pouvez le constater dans ce qui suit, `Window1` est une fenêtre de type `NavigationWindow`.

```
<NavigationWindow x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Mon premier wizard"
 Source="Page1.xaml"
 Width="280"
 Height="160"
 />
```

Page1 est automatiquement chargé dans la fenêtre navigable.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
>
   <StackPanel>
      <RadioButton Name="radA" MaxWidth="50"
       Margin="5,5,5,5">
         A.
      </RadioButton>
      <RadioButton Name="radB" MaxWidth="50"
       Margin="5,5,5,5">
         B.
      </RadioButton>
      <WrapPanel HorizontalAlignment="Center">
         <Button Width="80" Margin="5,5,5,5"
          Click="Prec" >
            Retour
         </Button>
         <Button Width="80" Margin="5,5,5,5"
          Click="Suiv" >
            Suivant
         </Button>
      </WrapPanel>
   </StackPanel>
</Page>

Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
```

```
' Interaction logic for Page1.xaml
Partial Public Class Page1
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Prec(ByVal sender As Object _
                  , ByVal e As RoutedEventArgs)
        Me.NavigationService.GoBack()
    End Sub

    Public Sub Suiv(ByVal sender As Object _
                  , ByVal e As RoutedEventArgs)
        If Me.radA.IsChecked Then
            Me.NavigationService.Navigate( _
             New Uri("Page1A.xaml", UriKind.Relative))
        Else
            Me.NavigationService.Navigate( _
            New Uri("Page1B.xaml", UriKind.Relative))
        End If
    End Sub

End Class
```



◄ **Figure 5-27 :** Première page du Wizard

Notez que le bouton **Retour** déclenche la méthode `GoBack` de la fenêtre de navigation tout comme si vous aviez appuyé sur la flèche Retour Arrière.

Si nous choisissons l'option *A*, la méthode nous fait naviguer vers la page *Page1A.xaml*.

```
<Page x:Class="Page1A"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1A"
 >
   <StackPanel>
```

```
      <RadioButton Name="rad1" MaxWidth="50"
       Margin="5,5,5,5">
         1.
      </RadioButton>
      <RadioButton Name="rad2" MaxWidth="50"
       Margin="5,5,5,5">
         2.
      </RadioButton>
      <WrapPanel HorizontalAlignment="Center">
         <Button Width="80"  Margin="5,5,5,5"
          Click="Prec">
            Retour
         </Button>
         <Button Width="80" Margin="5,5,5,5"
          Click="Suiv">
            Suivant
         </Button>
      </WrapPanel>
   </StackPanel>
</Page>

Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes

Partial Public Class Page1A
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Prec(ByVal sender As Object _
                  , ByVal e As RoutedEventArgs)
        Me.NavigationService.GoBack()
    End Sub

    Public Sub Suiv(ByVal sender As Object _
                , ByVal e As RoutedEventArgs)
        If Me.rad1.IsChecked Then
            Me.NavigationService.Navigate( _
             New Uri("Page1A1.xaml", UriKind.Relative))
        Else
            Me.NavigationService.Navigate( _
```

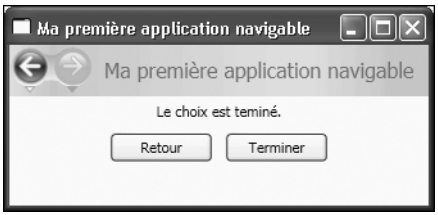
```
            New Uri("Page1A2.xaml", UriKind.Relative))
        End If
    End Sub

End Class
```

◄ **Figure 5-28 :** Deuxième page du Wizard

Cette fois, le choix 2 nous conduit vers une dernière page.

```
<Page x:Class="Page1A2"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1A2"
 >
   <StackPanel>
      <Label HorizontalContentAlignment="Center">
         Le choix est teminé.
      </Label>
      <WrapPanel HorizontalAlignment="Center">
         <Button Width="80"  Margin="5,5,5,5"
          Click="Prec">
            Retour
         </Button>
         <Button Width="80" Margin="5,5,5,5"
          Click="Terminer">
            Terminer
         </Button>
      </WrapPanel>
   </StackPanel>
</Page>

Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Navigation
Imports System.Windows.Shapes
```

```
' Interaction logic for Page1.xaml
Partial Public Class Page1A2
    Inherits Page

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Prec(ByVal sender As Object _
                  , ByVal e As RoutedEventArgs)
        Me.NavigationService.GoBack()
    End Sub

    Public Sub Terminer(ByVal sender As Object _
                  , ByVal e As RoutedEventArgs)
        DirectCast(Me.Parent, NavigationWindow).Close()

    End Sub

End Class
```



◄ **Figure 5-29 :** Troisième page du Wizard

Pour fermer la fenêtre, il est nécessaire de convertir la propriété `Parent` en une `NavigationWindow` que nous savons qu'elle est.

Si nous demandons le retour en arrière, les flèches avant et arrière deviennent accessibles.

◄ **Figure 5-30 :** Retour à la deuxième page du Wizard

Remarque

Boutons de navigation

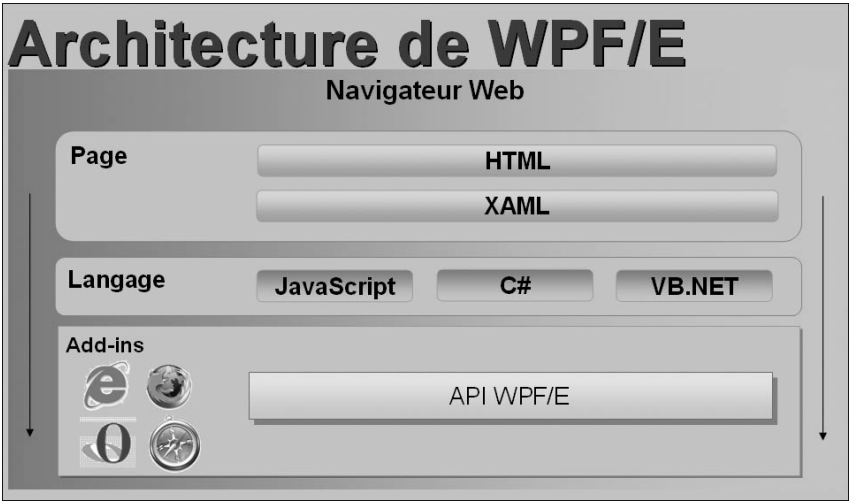
La fenêtre ne rend accessibles les boutons de navigation que si la navigation dans ce sens est possible. Vous pouvez également connaître cette information en utilisant les propriétés `CanGoBack` et `CanGoForward`.

5.6 Les applications avec WPF/E

WPF/E, abréviation pour *Windows Presentation Fundation for Everywhere*, a pour but comme son nom l'indique de pouvoir exécuter des applications WPF sur toutes les plates-formes. Les informations disponibles sur ce sujet sont encore très fragmentaires. Toutefois, on peut d'ores et déjà dire que ces applications vont généralement être exécutées dans un browser. Toutefois, il ne faut pas confondre application WBA et application WPF/E. En effet, en dehors des contraintes de sécurité, les WBA disposent de toute la puissance de WPF alors que WPF/E ne sera qu'un sous-ensemble de WPF. Alors, pourquoi WPF/E ? WPF/E pourra s'exécuter sur des plates-formes où WPF n'est pas installé. En fait, WPF/E, à l'instar de Flash, sera installé sur le poste client comme un Add-in du browser. Cet Add-in ne devrait pas peser plus de deux méga-octets. C'est pourquoi une partie du potentiel de WPF doit être amputé. Si, actuellement, il n'est pas encore possible de savoir exactement ce qui sera exactement inclus, il est en revanche déjà acquis que la 3D ne sera pas présente. En ce qui concerne la syntaxe XAML, pas de soucis, elle est identique mais bien sûr limitée aux fonctionnalités présentes.

WPF/E sera disponible pour Internet Explorer, Mozilla, Firefox, Opera et Safari. Au niveau du système d'exploitation, Windows (à partir de 2000) sera bien sûr supporté ainsi que Mac OS X 10. Pour Linux et Solaris, l'espoir demeure mais sans aucune certitude. Il en est de même pour les anciennes versions de Windows (Windows 95, 98 et Me). Des versions pour les PC de poche et autres téléphones portables utilisant les systèmes Windows sont également prévues.

Normalement, une première préversion doit voir le jour dans le courant du troisième trimestre 2006 alors que la version définitive est attendue pour le premier semestre 2006. En ce qui concerne les unités légères, il faudra attendre le second semestre 2007.



▲ **Figure 5-31 :** Architecture de WPF/E

Comme vous pouvez le constater, XAML pourra interagir avec le JavaScript mais également exécuter du code intermédiaire (compilé) .NET. Ce dernier pourra être exécuté directement par l'Add-in. Le code XAML pourra être intégré à la page HTML ou utilisé comme une ressource externe.

Voici tout d'abord comment devrait se présenter une page avec une application WPF/E en ressource externe.

```
<html>
   <body>
      <object id="wpfehost" size="…">
         <param name="source" value="monAppliction.wpfe"/>
…
   </body>
</html>
```

L'application WPF/E contiendra le XAML compressé (*baml*) mais aussi le code IL (.NET compilé) et les ressources (images, média...).

Maintenant, voyons comment devrait se présenter une application WPF/E intégrée à la page HTML.

```
<html>
  <head>
    <!–Script XAML-->
    <script id="monScriptXaml" type="text/xaml">
      <? Xml version="1.0" ?>
```

```
      <Canvas
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
        ...
      </Canvas>
    </script>
  </head>
  <body>
    ...
    <object id="wpfe" classid="..."
     codebase="xcpctrl.cab#version=0,0,3,0"
     height="300" width="400">
      <param name="SourceElement" value="monScriptXaml"/>
      <param name="WindowslessMode" value="True"/>
      <param name="BackgroundColor value="#00000000"/>
    </object>
    ...
  </body>
</html>
```

Comme vous l'avez vu dans le diagramme d'architecture, vous pouvez manipuler des objets WPF/E depuis le code JavaScript. Voici en quelques lignes comment faire.

```
<script type="text/javascript">
   void Fuction maFonction()
   {
      var wpfeObject = document.getElementById("wpfe") ;
      var ctrl = wpfeObject.FindName("LeNomDeMonControleDansXAML") ;
      ctrl.SetValue("NomDeLaPropriété", valeur) ;
   }
</script>
```

Vous pouvez par ce moyen créer des pages qui utilisent à la fois le HTML, le JavaScript et WPF/E au travers de XAML et si nécessaire de code .NET. Ces technologies peuvent non seulement cohabiter mais également interagir.

5.7 Checklist

Dans ce chapitre essentiellement dirigé sur la gestion d'une application, nous avons vu :

- comment est composée une application Windows et comment la réaliser ;
- comment réaliser une application à laquelle on accédera depuis un navigateur web (application WBA) ;
- comment utiliser `PageFunction` ;

- comment réaliser une page maître pour naviguer dans l'application ;
- comment gérer la sécurité pour les applications WBA ;
- comment transformer simplement une application WBA en application Windows ;
- comment réaliser un assistant (Wizard) ;
- comment gérer les événements dans XAML ;
- les bases du futur modèle d'application WPF/E.

Les menus

Créer un menu 170
Créer un menu contextuel 178
Créer une barre d'outils 183
Checklist 189

6.1 Créer un menu

Le menu principal

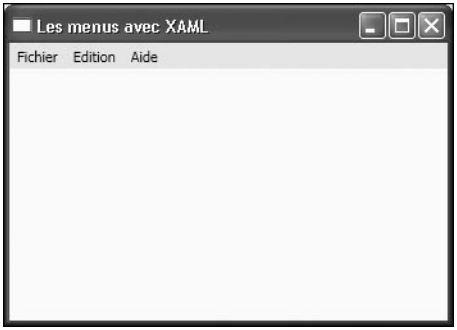
Traditionnellement, le menu principal se présente horizontalement en haut de fenêtre sur un fond gris clair. Pour créer notre menu, nous allons utiliser les classes `Menu` et `MenuItem`. La propriété `VerticalAlignment` va nous permettre de placer le menu sous la barre de titre comme souhaité.

```
<Window x:Class="AvalonMenu.Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec Avalon">
   <Menu VerticalAlignment="Top" Height="20">
      <MenuItem Header="Fichier"/>
      <MenuItem Header="Edition"/>
      <MenuItem Header="Aide"/>
   </Menu>
</Window>
```

Astuce

Hauteur du menu

La propriété `Height` est obligatoire sinon le menu occupera toute la hauteur disponible dans notre fenêtre.



◄ **Figure 6-1 :** Le menu principal

Dans l'exemple précédent, la fenêtre principale ne pourra contenir que le menu. C'est toutefois le cas de beaucoup d'applications et essentiellement des applications MDI. Si tel n'est pas votre objectif, la solution est simple et déjà connue. Il nous suffit de placer le menu dans un conteneur tel qu'une grille, un `StackPanel` ou encore un `DockPanel`.

```
<Window
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 >
   <DockPanel>
      <Menu VerticalAlignment="Top" Height="20"
       DockPanel.Dock="Top">
         <MenuItem Header="Fichier"/>
         <MenuItem Header="Edition"/>
         <MenuItem Header="Aide"/>
      </Menu>
   </DockPanel>
</Window>
```

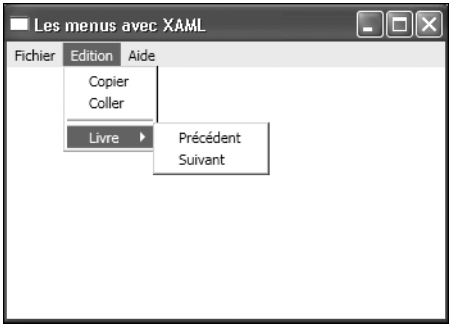
Le résultat sera identique, à la différence près qu'il est maintenant possible de placer d'autres contrôles ou d'autres conteneurs dans la fenêtre.

Les sous-menus

Pour créer un sous-menu, un menu vertical, il suffit de définir un ou des `MenuItem` dans le nœud `MenuItem` du menu principal. Chaque nouveau `MenuItem` peut à son tour contenir d'autres `MenuItem` fils.

```
<Window
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 >
   <DockPanel>
      <Menu VerticalAlignment="Top" Height="20"
       DockPanel.Dock="Top">
         <MenuItem Header="Fichier">
            <MenuItem Header="Ouvrir"/>
            <MenuItem Header="Fermer"/>
         </MenuItem>
         <MenuItem Header="Edition">
            <MenuItem Header="Copier"/>
            <MenuItem Header="Coller"/>
            <Separator />
            <MenuItem Header="Livre">
               <MenuItem Header="Précédent"/>
               <MenuItem Header="Suivant"/>
            </MenuItem>
         </MenuItem>
         <MenuItem Header="Aide"/>
      </Menu>
```

```
    </DockPanel>
</Window>
```



◄ **Figure 6-2 :** Les sous-menus

Remarque

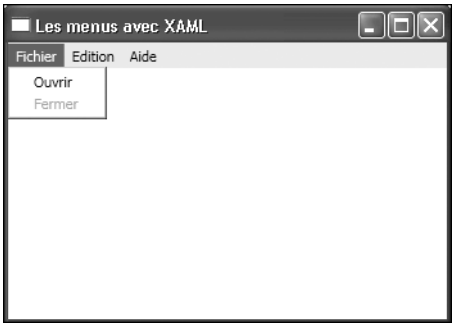
Placer des séparations dans le menu

Notez l'utilisation de la balise `Separator` pour créer un séparateur.

Rendre un élément du menu inactif

Pour rendre un élément d'un menu inactif (grisé), vous devez assigner la valeur `False` à l'attribut `IsEnabled`.

```
<MenuItem Header="Fichier">
    <MenuItem Header="Ouvrir"/>
    <MenuItem Header="Fermer" IsEnabled="False"/>
</MenuItem>
```

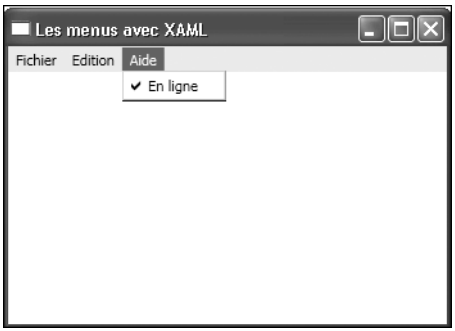


◄ **Figure 6-3 :** Les sous-menus

Cocher un élément du menu

Pour cocher un élément d'un menu, utilisez la propriété `IsChecked`. Si elle est `True`, l'élément du menu correspondant sera coché.

```
<MenuItem Header="Aide">
   <MenuItem Header="En ligne" IsChecked="True"/>
</MenuItem>
```



◄ **Figure 6-4 :** Les sous-menus

Associer une action à un menu

Pour associer une méthode au clic sur un élément du menu, il suffit d'assigner le nom de la méthode à l'attribut `Click` de l'élément correspondant. Reprenons l'exemple précédent complet et adaptons-le pour rendre le menu **Fermer** disponible quand on clique sur **Ouvrir** et rendre en revanche ce dernier indisponible. Nous ferons l'inverse avec **Fermer**. Dans le même ordre d'idée, nous supprimerons ou activerons la coche lors du clic sur le menu en ligne.

Attention

Réaliser soi-même l'exemple

Si vous souhaitez reproduire cet exemple, vous devrez créer un projet dans Visual Studio.

Le code XAML devient :

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 >
   <DockPanel>
```

```
      <Menu VerticalAlignment="Top" Height="20"
       DockPanel.Dock="Top">
         <MenuItem Header="Fichier">
            <MenuItem Name="mnuOuvrir" Header="Ouvrir"
             Click="Click_Ouvrir"/>
            <MenuItem Name="mnuFermer" Header="Fermer"
             IsEnabled="False" Click="Click_Fermer"/>
         </MenuItem>
         <MenuItem Header="Edition">
            <MenuItem Header="Copier"/>
            <MenuItem Header="Coller"/>
            <Separator />
            <MenuItem Header="Livre">
               <MenuItem Header="Précédent"/>
               <MenuItem Header="Suivant"/>
            </MenuItem>
         </MenuItem>
         <MenuItem Header="Aide">
            <MenuItem Name="mnuEnLigne" Header="En ligne"
             IsChecked="True" Click="Click_EnLigne"/>
         </MenuItem>
      </Menu>
   </DockPanel>
</Window>
```

Remarque

Importance des noms

Jusqu'à maintenant, dans les menus, je n'avais pas pris la peine de donner des noms. Comme vous pouvez le constater, cela se révèle rapidement nécessaire.

Les méthodes qui seront appelées lors des événements `Click` respectifs sont définies dans le code VB.NET associé.

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Shapes

' Interaction logic for Window1.xaml
Partial Public Class Window1
    Inherits Window
```

```
    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Click_Ouvrir(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = True
        Me.mnuOuvrir.IsEnabled = False
    End Sub

    Public Sub Click_Fermer(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = False
        Me.mnuOuvrir.IsEnabled = True
    End Sub

    Public Sub Click_EnLigne(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuEnLigne.IsChecked = _
                        Not Me.mnuEnLigne.IsChecked
    End Sub

End Class
```

La syntaxe d'écriture des méthodes appelées par les événements est tout à fait conforme à ce que nous avons vu dans le chapitre réservé à ce sujet.

Attention

Problème avec l'IntelliSense

Actuellement, l'IntelliSense ne reconnaît pas directement les noms que nous avons ajoutés. Recompilez votre projet et ceux-ci apparaîtront.

Pour changer notre menu, nous devons simplement modifier les propriétés `IsChecked` et `IsEnabled` qui correspondent aux attributs du même nom.

Astuce

Inverser une valeur

Nous ne devons pas connaître l'état de `IsChecked` pour cocher ou décocher l'élément **En ligne** du menu. Il suffit d'inverser la valeur pour obtenir le résultat souhaité, ce qui se fait simplement en réassignant la valeur à elle-même et en la précédant de `Not`.

Rendre le menu dynamique

Plutôt qu'activer ou désactiver certaines options du menu, vous souhaiterez certainement rendre ce menu dynamique en fonction des événements déclenchés. Par exemple, le menu **Edition** est superflu si le fichier n'a pas été ouvert.

Pour pouvoir accéder à toutes les fonctionnalités d'un menu depuis le code .NET, il suffit de manipuler l'objet de la classe `Menu`, qui est forcément instancié. Pour y accéder facilement, il doit être nommé et, oups, nous ne l'avons pas fait ! Réparons vite cet oubli.

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 >
   <DockPanel>
      <Menu Name="mnuMain" VerticalAlignment="Top"
       Height="20" DockPanel.Dock="Top">
        <MenuItem Name="mnuFichier" Header="Fichier">
          <MenuItem Name="mnuOuvrir" Header="Ouvrir"
           Click="Click_Ouvrir"/>
          <MenuItem Name="mnuFermer" Header="Fermer"
           IsEnabled="False" Click="Click_Fermer"/>
        </MenuItem>
        <MenuItem Name="mnuEdition" Header="Edition"
         Visibility="Collapsed">
          <MenuItem Name="mnuCopier" Header="Copier"/>
          <MenuItem Name="mnuColler" Header="Coller"/>
          <Separator />
          <MenuItem Name="mnuLivre" Header="Livre">
            <MenuItem Name="mnuPrec" Header="Précédent"/>
            <MenuItem Name="mnuSuiv" Header="Suivant"/>
          </MenuItem>
        </MenuItem>
        <MenuItem Header="Aide">
          <MenuItem Name="mnuEnLigne" Header="En ligne"
           IsChecked="True" Click="Click_EnLigne"/>
        </MenuItem>
      </Menu>
   </DockPanel>
</Window>
```

Comme vous pouvez le constater, tous les éléments du menu sont maintenant nommés, ce qui nous permettra d'accéder à n'importe lequel des éléments.

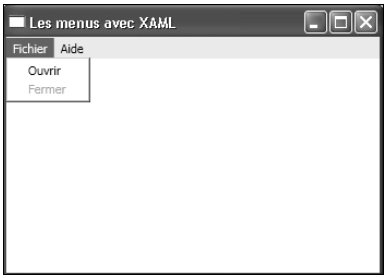
Un autre attribut important a également été ajouté. L'attribut `Visibility` permet de cacher ou d'afficher un élément. Il peut prendre trois valeurs différentes.

- `visible` : l'élément est affiché.
- `hidden` : l'élément n'est pas affiché mais sa place est réservée.
- `collapsed` : l'élément n'est pas affiché et sa place n'est pas réservée, ce qui provoque un déplacement des autres éléments du menu.

Pour des raisons évidentes, utilisez `Collapsed` plutôt que `Hidden`.

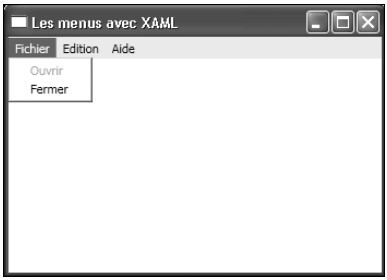
```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Shapes

' Interaction logic for Window1.xaml
Partial Public Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub Click_Ouvrir(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = True
        Me.mnuOuvrir.IsEnabled = False
        Me.mnuEdition.Visibility = Windows.Visibility.Visible
    End Sub

    Public Sub Click_Fermer(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = False
        Me.mnuOuvrir.IsEnabled = True
        Me.mnuEdition.Visibility = _
                            Windows.Visibility.Collapsed
    End Sub

    Public Sub Click_EnLigne(ByVal sender As Object _
, ByVal e As RoutedEventArgs)
        Me.mnuEnLigne.IsChecked = Not Me.mnuEnLigne.IsChecked
    End Sub

End Class
```

L'écran au démarrage se présente comme ceci.



◀ **Figure 6-5** : Menu dynamique

Mais, après avoir cliqué sur **Ouvrir**, le menu **Edition** est ajouté.



◀ **Figure 6-6** : Menu dynamique

Grâce à l'utilisation de la propriété `Visibility`, le code .NET est très simple. Cela nécessite de penser complètement le menu dans le module de base. Si vous êtes amené à ajouter un menu de manière totalement dynamique, c'est-à-dire sans qu'il soit prévu initialement, vous devrez le faire uniquement au moyen de code .NET.

6.2 Créer un menu contextuel

Comme son nom l'indique, le menu contextuel est destiné à offrir un menu spécifique à l'élément auquel il est attaché. L'accès à ce menu se fait grâce au clic droit de la souris. Il est de plus en plus couramment utilisé dans toutes les applications et plus seulement les applications professionnelles. D'autant que sa programmation est en définitive relativement aisée.

Pour illustrer le menu contextuel, nous allons associer l'exemple précédent avec l'exemple repris dans le paragraphe sur le `Canvas`. Le menu contextuel est associé au `Canvas`.

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

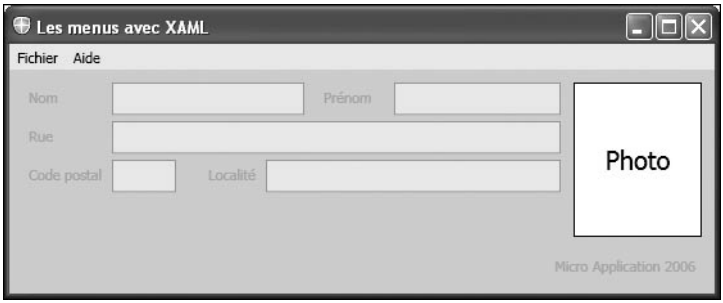
```
Title="Les menus avec XAML"
>
  <DockPanel>
     <Menu Name="mnuMain" VerticalAlignment="Top"
       Height="20" DockPanel.Dock="Top">
        <MenuItem Name="mnuFichier" Header="Fichier">
           <MenuItem Name="mnuOuvrir" Header="Ouvrir"
             Click="Click_Ouvrir"/>
           <MenuItem Name="mnuFermer" Header="Fermer"
             IsEnabled="False" Click="Click_Fermer"/>
        </MenuItem>
        <MenuItem Name="mnuEdition" Header="Edition"
          Visibility="Collapsed">
           <MenuItem Name="mnuCopier" Header="Copier"/>
           <MenuItem Name="mnuColler" Header="Coller"/>
           <Separator />
           <MenuItem Name="mnuLivre" Header="Livre">
              <MenuItem Name="mnuPrec"
               Header="Précédent"
               Click="Click_Prev"/>
              <MenuItem Name="mnuSuiv"
               Header="Suivant"
               Click="Click_Next"/>
           </MenuItem>
        </MenuItem>
        <MenuItem Header="Aide">
           <MenuItem Name="mnuEnLigne" Header="En ligne"
            IsChecked="True" Click="Click_EnLigne"/>
        </MenuItem>
     </Menu>
     <Canvas Name="frmData" IsEnabled="False"
      Background="LightBlue" DockPanel.Dock="Bottom">
        <Canvas.ContextMenu>
           <ContextMenu>
              <MenuItem Header="Précédent"
               Click="Click_Prev"/>
              <MenuItem Header="Suivant"
               Click="Click_Next"/>
           </ContextMenu>
        </Canvas.ContextMenu>
        <Label Name="lblNom" Canvas.Top="10"
         Canvas.Left="10">
           Nom
        </Label>
        <TextBox Name="txtNom"  Canvas.Top="10"
         Canvas.Left="80" Width="150" MaxLength="30"
         CharacterCasing="Upper" />
        <Label Name="lblPrenom" Canvas.Top="10"
         Canvas.Left="240">
           Prénom
```

```
        </Label>
        <TextBox Name="txtPrenom"  Canvas.Top="10"
         Canvas.Left="300" Width="130" MaxLength="30"/>
        <Label Name="lblAdr" Canvas.Top="40"
         Canvas.Left="10">
           Rue
        </Label>
        <TextBox Name="txtAdr"  Canvas.Top="40"
         Canvas.Left="80" Width="350"
         MaxLength="80"/>
        <Label Name="lblCP" Canvas.Top="70"
         Canvas.Left="10">
           Code postal
        </Label>
        <TextBox Name="txtCP"  Canvas.Top="70"
         Canvas.Left="80"
         Width="50"  MaxLength="5"/>
        <Label Name="lblLocalite" Canvas.Top="70"
         Canvas.Left="150">
           Localité
        </Label>
        <TextBox Name="txtLocalite"  Canvas.Top="70"
         Canvas.Left="200" Width="230"  MaxLength="50"/>
        <Border Width="100" Height="120"
         BorderThickness="1"
         Background="White" BorderBrush="Black"
         Canvas.Top="10" Canvas.Right="10">
           <TextBlock Name="blkPhoto"
            VerticalAlignment="Center"
            HorizontalAlignment="Center" FontSize="20">
              Photo
           </TextBlock>
        </Border>
        <Label Name="lblCopyright" Canvas.Bottom="10"
         Canvas.Right="10"
         Content="Micro Application 2006" />
     </Canvas>
   </DockPanel>
</Window>
```

Le code VB.NET associé est modifié comme ceci.

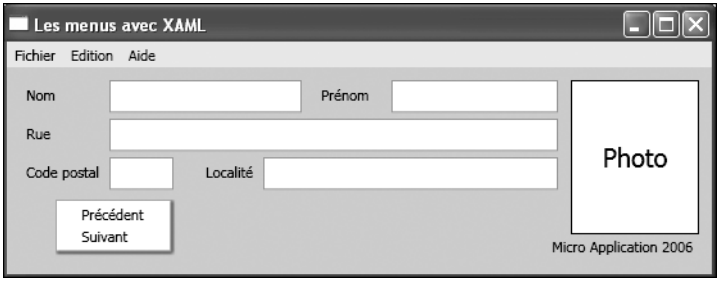
```
' Interaction logic for Window1.xaml
Partial Public Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub
```

```
    Public Sub Click_Ouvrir(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = True
        Me.mnuOuvrir.IsEnabled = False
        Me.mnuEdition.Visibility = Windows.Visibility.Visible
        Me.frmData.IsEnabled = True
    End Sub

    Public Sub Click_Fermer(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        Me.mnuFermer.IsEnabled = False
        Me.mnuOuvrir.IsEnabled = True
        Me.mnuEdition.Visibility = Windows.Visibility.Collapsed
        Me.frmData.IsEnabled = False
    End Sub

    Public Sub Click_EnLigne(ByVal sender As Object _
                     , ByVal e As RoutedEventArgs)
        Me.mnuEnLigne.IsChecked = Not Me.mnuEnLigne.IsChecked
    End Sub

    Public Sub Click_Prev(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        ' Code pour précedent
    End Sub

    Public Sub Click_Next(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        ' Code pour suivant
    End Sub

End Class
```



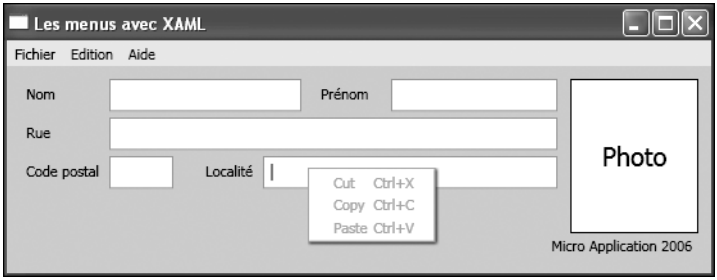
▲ **Figure 6-7** : Le menu contextuel dans un environnement inactif

Si vous essayez d'activer le menu contextuel, rien ne se passe. Le menu n'est pas accessible car le canevas est désactivé. L'activation du canevas est réalisée dans la méthode `Click_Ouvrir`. Après avoir effectué **Fichier**, **Ouvrir**, le menu contextuel est maintenant actif.



▲ **Figure 6-8 :** Le menu contextuel

Toutefois, les boîtes de texte conservent leur menu contextuel par défaut.

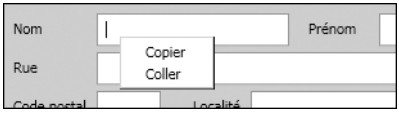


▲ **Figure 6-9 :** Le menu contextuel par défaut

Si vous désirez les remplacer, il faudra créer un autre menu contextuel.

Remplacez le premier `TextBox` par celui-ci et il possédera son propre menu contextuel. Évidemment, dans l'exemple, il est très simplifié. Aucune action ne lui est associée mais cela, vous savez déjà comment le résoudre.

```
<TextBox Name="txtNom"  Canvas.Top="10" Canvas.Left="80"
   Width="150" MaxLength="30" CharacterCasing="Upper">
   <TextBox.ContextMenu>
      <ContextMenu>
         <MenuItem Header="Copier"/>
         <MenuItem Header="Coller"/>
      </ContextMenu>
   </TextBox.ContextMenu>
</TextBox>
```



◄ **Figure 6-10 :** Un menu contextuel dans une boîte de texte

Renvoi *Vous trouverez page 132 comment partager un menu contextuel entre plusieurs éléments de l'écran.*

6.3 Créer une barre d'outils

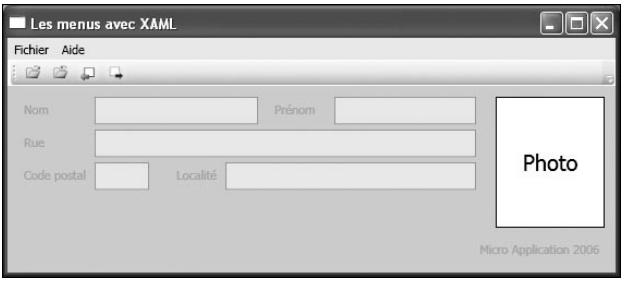
Une barre d'outils statique

La barre d'outils est créée grâce à un objet de la classe `ToolBar`. Pour placer les éléments dans la barre d'outils, il n'y a qu'à ajouter au sein du nœud `ToolBar` les contrôles que vous souhaitez voir apparaître. Généralement, une barre d'outils est principalement composée de boutons.

En ajoutant le code suivant derrière la balise fermante `Menu` de notre exemple précédent, nous obtiendrons directement une barre d'outils tout à fait fonctionnelle.

```
<ToolBar DockPanel.Dock="Top" Height="24">
   <Button ToolTip="Ouvrir" Click="Click_Ouvrir">
      <Image>
         <Image.Source>
            <Binding Source="open.bmp"/>
         </Image.Source>
      </Image>
   </Button>
   <Button ToolTip="Fermer" Click="Click_Fermer">
      <Image>
         <Image.Source>
            <Binding Source="close.bmp"/>
         </Image.Source>
      </Image>
   </Button>
   <Button ToolTip="Precedent" Click="Click_Prev">
      <Image>
         <Image.Source>
            <Binding Source="precedent.bmp"/>
         </Image.Source>
      </Image>
   </Button>
   <Button ToolTip="Suivant" Click="Click_Next">
      <Image>
         <Image.Source>
            <Binding Source="suivant.bmp"/>
```

```
        </Image.Source>
      </Image>
   </Button>
</ToolBar>
```



▲ **Figure 6-11 :** Une simple barre d'outils

Remarque

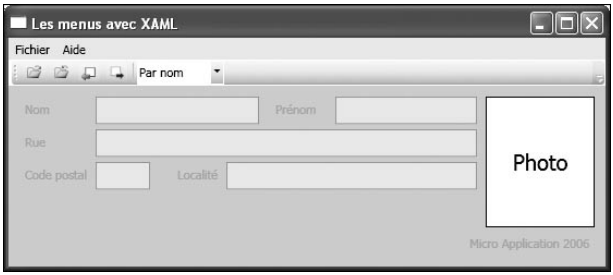
Positionnement de la barre d'outils

Nous avons ici utilisé un `DockPanel` pour placer la barre d'outils et le menu, mais nous aurions tout aussi bien pu utiliser une grille, un `StackPanel` ou même un `WrapPanel`. Le canevas est quant à lui moins approprié, mais pourquoi pas !

La barre d'outils peut contenir d'autres contrôles que le bouton. Par exemple une `ComboBox`.

```
<ToolBar DockPanel.Dock="Top" Height="24">
   <Button ToolTip="Ouvrir" Click="Click_Ouvrir">
      <Image>
         <Image.Source>
            <Binding Source="open.bmp"/>
         </Image.Source>
      </Image>
   </Button>
   <Button ToolTip="Fermer" Click="Click_Fermer">
      <Image>
         <Image.Source>
            <Binding Source="close.bmp"/>
         </Image.Source>
      </Image>
   </Button>
   <Button ToolTip="Precedent">
      <Image>
         <Image.Source>
```

```
            <Binding Source="precedent.bmp"/>
          </Image.Source>
       </Image>
    </Button>
    <Button ToolTip="Suivant">
       <Image>
          <Image.Source>
             <Binding Source="suivant.bmp"/>
          </Image.Source>
       </Image>
    </Button>
    <Separator/>
    <ComboBox Width="80">
       <ComboBoxItem IsSelected="True">
          Par nom
       </ComboBoxItem>
       <ComboBoxItem>
          Par localité
       </ComboBoxItem>
    </ComboBox>
</ToolBar>
```



▲ **Figure 6-12 :** Barre d'outils avec ComboBox

Remarque

Séparateur

La `ComboBox` est précédée d'un séparateur. Il ne s'agit absolument pas d'une obligation. Vous pouvez placer des séparateurs où bon vous semble simplement en y plaçant la balise `Separator` comme dans le code ci-dessus.

Un ensemble de barres d'outils

Les applications modernes ne se contentent pas d'afficher une seule barre d'outils mais affichent bien un ensemble dans lequel vous pouvez déplacer ou

redimensionner à votre guise les différentes barres d'outils. Avec XAML, nous pouvons créer un conteneur dans lequel nous allons placer nos barres d'outils. Elles pourront être alors déplacées ou redimensionnées par l'utilisateur dans l'espace déterminé par le conteneur. Si l'utilisateur réduit trop la taille d'une barre d'outils, les éléments qui ne sont plus affichables sont automatiquement ajoutés dans une zone de dépassement et rendus accessibles *via* la petite flèche qui marque la fin de la barre d'outils.

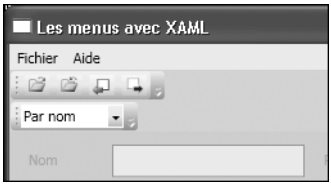
Pour créer un conteneur, vous devez utiliser la balise `ToolBarTray`.

```
<ToolBarTray DockPanel.Dock="Top" IsLocked="False"
 Background="LightGray">
   <ToolBar Band="0" Height="24">
      <Button ToolTip="Ouvrir" Click="Click_Ouvrir">
         <Image>
            <Image.Source>
               <Binding Source="open.bmp"/>
            </Image.Source>
         </Image>
      </Button>
      <Button ToolTip="Fermer" Click="Click_Fermer">
         <Image>
            <Image.Source>
               <Binding Source="close.bmp"/>
            </Image.Source>
         </Image>
      </Button>
      <Button ToolTip="Precedent">
         <Image>
            <Image.Source>
               <Binding Source="precedent.bmp"/>
            </Image.Source>
         </Image>
      </Button>
      <Button ToolTip="Suivant">
         <Image>
            <Image.Source>
               <Binding Source="suivant.bmp"/>
            </Image.Source>
         </Image>
      </Button>
      </ToolBar>
      <ToolBar Band="1" Height="24">
      <ComboBox Width="80">
         <ComboBoxItem IsSelected="True">
            Par nom
         </ComboBoxItem>
         <ComboBoxItem>
            Par localité
```

```
            </ComboBoxItem>
        </ComboBox>
    </ToolBar>
</ToolBarTray>
```

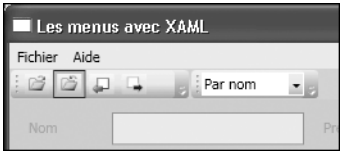
Vous pouvez constater que la barre d'outils précédemment créée a été scindée en deux au niveau du séparateur. C'est maintenant la `ToolBarTray` qui reçoit l'attribut `DockPanel.Dock`. Pour éviter un fond blanc, l'attribut `Background` reçoit la valeur `LightGray`.

L'attribut `Band` dans la balise `ToolBar` permet de fixer le numéro de ligne dans lequel vous désirez voir apparaître la barre d'outils.



◄ **Figure 6-13 :** Des barres d'outils mobiles

L'utilisateur peut déplacer les barres d'outils, soit pour les rassembler sur une même ligne, soit pour les inverser. Il ne pourra en revanche pas les décaler vers la gauche en laissant un espace entre les barres d'outils.



◄ **Figure 6-14 :** Des barres d'outils mobiles

Astuce

Bloquer les barres d'outils

La propriété `IsLocked` permet de spécifier si les barres d'outils contenues sont mobiles ou non. Il est possible de le spécifier individuellement pour chaque barre en introduisant dans la balise de la barre concernée l'attribut `ToolBarTray.IsLocked`.

Si vous souhaitez afficher la barre d'outils à gauche plutôt qu'en haut de l'écran, c'est très simple. Vous changez l'attribut `Orientation` et c'est pratiquement tout.

```
<ToolBarTray DockPanel.Dock="Left" IsLocked="False"
 Background="LightGray" Orientation="Vertical">
```

En effet, il est également judicieux de changer le docking, sans quoi le résultat ne sera pas celui espéré.

Attention

Ordre des lignes

Vu le changement d'orientation, les lignes définies par l'attribut `Band` deviennent des colonnes. C'est pourquoi il est préférable dans notre exemple de fixer la valeur à 0 pour les deux barres d'outils.

```
<ToolBar Band="0">
```

Attention

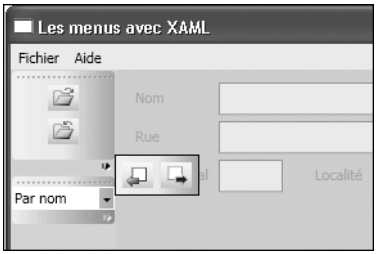
Hauteur des lignes

Pour les mêmes raisons, la hauteur qui était fixée à 24 points doit être adaptée pour par exemple devenir automatique. Pour y arriver, retirez l'attribut `Height` de chaque balise `ToolBar`.

◄ **Figure 6-15 :** Des barres d'outils à gauche

Avec ces barres d'outils flottantes, nous devons nous pencher d'un peu plus près sur la zone de débordement. Si vous réduisez la taille de la première barre d'outils en montant la seconde, la zone de débordement indiquée par la flèche maintenant noire et non plus grise est activée. Malheureusement, si vous tentez de l'utiliser, l'icône cachée va bel et bien s'afficher mais en très grand. Pour éviter ce problème, il est prudent d'utiliser les attributs `MaxWidth` et `MaxHeigth` des images des différents boutons. Les autres contrôles éventuels doivent subir le même traitement.

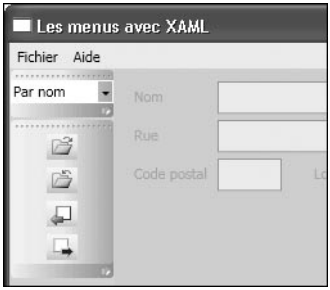
```
<Image MaxHeight="24" MaxWidth="24">
```



◄ **Figure 6-16 :** Zone de débordement de la barre d'outils

Si vous souhaitez changer l'ordre des barres d'outils sans déplacer tout le code, vous pouvez simplement utiliser l'attribut `BandIndex`. Dans l'exemple, assignez 1 à cet attribut pour la première barre d'outils.

```
ToolBar Band="0" BandIndex="1">
```



◄ **Figure 6-17 :** Ordre des barres d'outils

Attention

Numérotation

Pour rappel, la numérotation commence à 0.

6.4 Checklist

Dans ce chapitre, nous avons vu comment créer et manipuler les menus et particulièrement comment :

- créer un menu d'application complet avec sous-menu et menus imbriqués ;
- rendre un menu dynamique ;
- créer un menu contextuel ;
- créer et manipuler une barre d'outils.

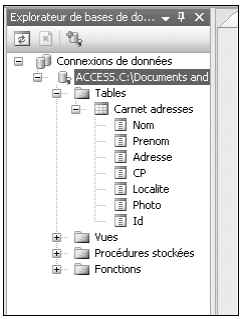
Lier les données à son interface utilisateur

Lier les données à un DataSet 192
Lier les données à un objet métier 203
Lier les données sans utiliser le code .NET 207
Checklist 218

Outre la possibilité d'assigner vous-même les valeurs aux différents contrôles, vous aurez certainement envie de mettre en place des techniques comme la liaison aux données (**Data Binding**) pour lier vos contrôles avec les données qui doivent y être reprises. Cette technique n'est pas neuve mais nous allons voir comment la mettre en œuvre avec XAML.

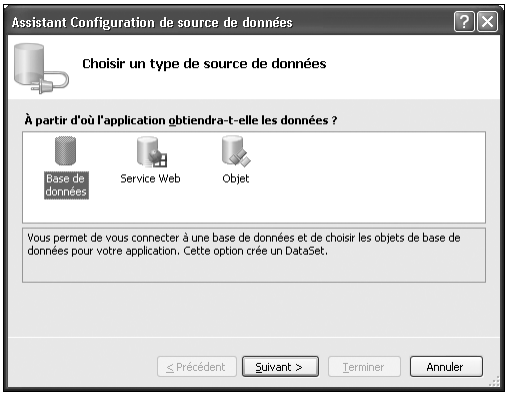
7.1 Lier les données à un DataSet

Tout d'abord, nous devons disposer d'un `DataSet`. Pour que vous puissiez réaliser vous-même l'exercice sans devoir installer un gestionnaire de bases de données quelconque, le choix de réaliser un `DataSet` vers une base de données MS-Access dont vous pouvez voir la structure dans l'écran ci-dessous semble le plus judicieux.



◄ **Figure 7-1 :** Structure de la table

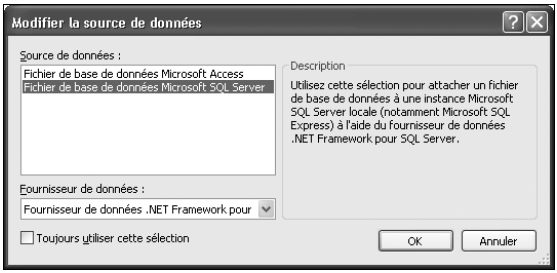
Une fois la base de données créée, nous devons l'ajouter à notre projet. Choisissez dans le menu de Visual Studio **Données...** - **Ajouter une nouvelle source de données...**



▲ **Figure 7-2 :** Ajouter une source de données

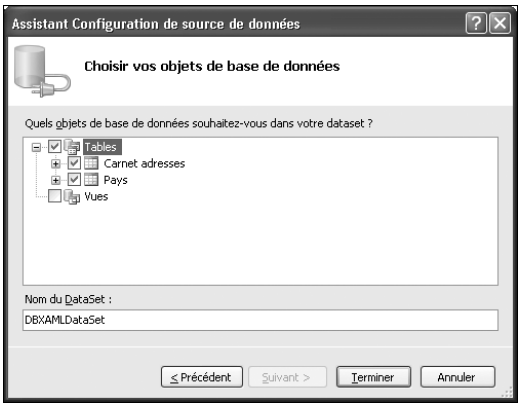
Si ce n'est déjà fait, choisissez l'option *Base de données* et cliquez sur **Suivant**.

Cliquez sur le bouton **Nouvelle connexion...** et ensuite sur le bouton **Modifier**.



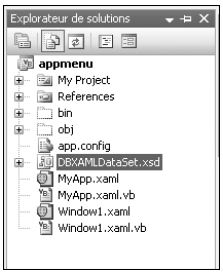
▲ **Figure 7-3 :** Modifier le type de base de données

Cliquez maintenant sur le bouton **Parcourir**, choisissez la base de données que vous venez de créer et cliquez sur OK puis sur **Suivant**.



▲ **Figure 7-4 :** Contenu à inclure dans le DataSet

Sélectionnez la table à inclure dans le `DataSet`.



◄ **Figure 7-5 :** Un DataSet fortement typé

Le DataSet est maintenant créé. Il ne nous reste qu'à l'inclure dans notre programme. Modifiez le code de *Window1.xaml.vb* pour refléter le code ci-dessous.

```
Private m_data As New DBXAMLDataSet
Private m_i As Integer

Public Sub New()
   InitializeComponent()
   Dim adapter As New _
   DBXAMLDataSetTableAdapters.Carnet_adressesTableAdapter
   adapter.Fill(m_data.Carnet_adresses)
End Sub
```

Une fois cette étape réalisée, nous devons maintenant associer le DataSet à notre fenêtre. Commençons par ajouter la gestion de l'événement Loaded de la fenêtre.

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 Loaded="Win_loaded"
 >
```

La méthode Win_loaded est appelée après le chargement de la fenêtre. Dans cette méthode, nous allons initialiser la source de données.

```
Private Sub Win_loaded(ByVal sender As Object _
                     , ByVal e As RoutedEventArgs)
   m_i = 0
   Me.DataContext = m_data.Carnet_adresses.Rows(m_i)
End Sub
```

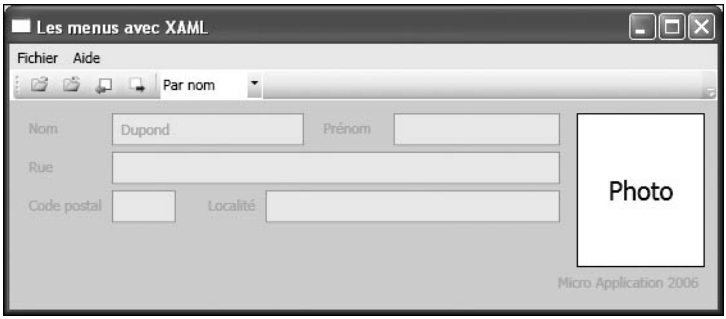
La liaison se fait en utilisant la propriété DataContext de la fenêtre.

Il reste maintenant à lier les champs avec les zones écran. Commençons par le nom.

```
<TextBox Name="txtNom"
 Canvas.Top="10" Canvas.Left="80"
 Width="150" MaxLength="30" CharacterCasing="Upper"
      Text ="{Binding Path=Nom}">
```

Pour réaliser la liaison, nous devons utiliser un objet de type `Binding`. L'objet doit être assigné à la propriété, qui doit être liée. Pour y arriver, nous avons utilisé une syntaxe un peu particulière.

Nous pouvons déjà exécuter l'application telle quelle.



▲ **Figure 7-6** : Première liaison

Le nom est correctement affiché à l'écran. Faisons de même avec les autres champs.

Pour vous aider à reconstruire l'exemple, vous trouverez ci-dessous le code complet que nous avons déjà discuté précédemment.

```
<Window x:Class="Window1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Les menus avec XAML"
 Loaded="Win_loaded"
 >
   <DockPanel>
      <Menu Name="mnuMain" VerticalAlignment="Top"
       Height="20" DockPanel.Dock="Top">
        <MenuItem Name="mnuFichier" Header="Fichier">
        <MenuItem Name="mnuOuvrir" Header="Ouvrir"
         Click="Click_Ouvrir"/>
        <MenuItem Name="mnuFermer" Header="Fermer"
         IsEnabled="False" Click="Click_Fermer"/>
        </MenuItem>
        <MenuItem Name="mnuEdition" Header="Edition"
         Visibility="Collapsed">
```

```
        <MenuItem Name="mnuCopier" Header="Copier"/>
        <MenuItem Name="mnuColler" Header="Coller"/>
        <Separator />
        <MenuItem Name="mnuContact" Header="Contact">
           <MenuItem Name="mnuPrec"
            Header="Précédent"
            Click="Click_Prev"/>
           <MenuItem Name="mnuSuiv"
            Header="Suivant"
            Click="Click_Next"/>
        </MenuItem>
     </MenuItem>
     <MenuItem Header="Aide">
        <MenuItem Name="mnuEnLigne" Header="En ligne"
         IsChecked="True" Click="Click_EnLigne"/>
     </MenuItem>
  </Menu>
  <ToolBarTray DockPanel.Dock="Left" IsLocked="False"
   Background="LightGray" Orientation="Vertical">
     <ToolBar Band="0">
        <Button ToolTip="Ouvrir" Click="Click_Ouvrir">
           <Image MaxHeight="24" MaxWidth="24">
              <Image.Source>
                 <Binding Source="open.bmp"/>
              </Image.Source>
           </Image>
        </Button>
        <Button ToolTip="Fermer" Click="Click_Fermer">
            <Image MaxHeight="24" MaxWidth="24">
               <Image.Source>
                  <Binding Source="close.bmp"/>
               </Image.Source>
            </Image>
         </Button>
         <Button ToolTip="Precedent" Click="Click_Prev">
            <Image MaxHeight="24" MaxWidth="24">
               <Image.Source>
                  <Binding Source="precedent.bmp"/>
               </Image.Source>
            </Image>
         </Button>
         <Button ToolTip="Suivant" Click="Click_Next">
            <Image MaxHeight="24" MaxWidth="24">
               <Image.Source>
                  <Binding Source=" suivant.bmp"/>
               </Image.Source>
            </Image>
         </Button>
      </ToolBar>
      <ToolBar Band="0">
```

```
        <ComboBox Width="80">
           <ComboBoxItem IsSelected="True">
              Par nom
           </ComboBoxItem>
           <ComboBoxItem>
              Par localité
           </ComboBoxItem>
        </ComboBox>
     </ToolBar>
  </ToolBarTray>
  <Canvas Name="frmData" IsEnabled="False"
   Background="LightBlue" DockPanel.Dock="Bottom">
     <Canvas.ContextMenu>
        <ContextMenu>
           <MenuItem Header="Précédent"
            Click="Click_Prev"/>
           <MenuItem Header="Suivant"
            Click="Click_Next"/>
        </ContextMenu>
        </Canvas.ContextMenu>
        <Label Name="lblNom" Canvas.Top="10"
         Canvas.Left="10">
           Nom
        </Label>
        <TextBox Name="txtNom" Canvas.Top="10"
         Canvas.Left="80" Width="150"
         MaxLength="30" CharacterCasing="Upper"
         Text ="{Binding Path=Nom}">
           <TextBox.ContextMenu>
              <ContextMenu>
                 <MenuItem Header="Copier"/>
                 <MenuItem Header="Coller"/>
              </ContextMenu>
           </TextBox.ContextMenu>
        </TextBox>
        <Label Name="lblPrenom" Canvas.Top="10"
         Canvas.Left="240">
            Prénom
        </Label>
```

C'est à partir d'ici que le code est effectivement modifié. Pour chacun des `TextBox` un objet de type `Binding` est assigné à la propriété `Text`.

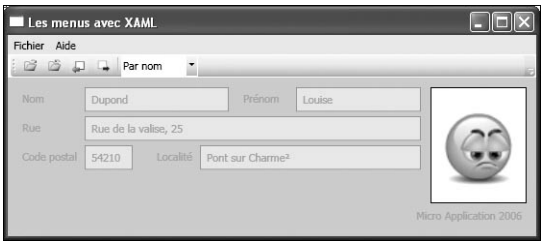
```
        <TextBox Name="txtPrenom"  Canvas.Top="10"
         Canvas.Left="300" Width="130" MaxLength="30"
         Text ="{Binding Path=Prenom}"/>
        <Label Name="lblAdr" Canvas.Top="40"
         Canvas.Left="10">
           Rue
```

```
        </Label>
        <TextBox Name="txtAdr"  Canvas.Top="40"
         Canvas.Left="80" Width="350"     MaxLength="80"
         Text ="{Binding Path=Adresse}"/>
        <Label Name="lblCP" Canvas.Top="70"
         Canvas.Left="10">
           Code postal
        </Label>
        <TextBox Name="txtCP"  Canvas.Top="70"
         Canvas.Left="80" Width="50"  MaxLength="5"
         Text ="{Binding Path=CP}"/>
        <Label Name="lblLocalite" Canvas.Top="70"
         Canvas.Left="150">
            Localité
        </Label>
        <TextBox Name="txtLocalite"  Canvas.Top="70"
         Canvas.Left="200" Width="230"  MaxLength="50"
         Text ="{Binding Path=Localite}"/>
        <Border Width="100" Height="120"
         BorderThickness="1"
         Background="White" BorderBrush="Black"
         Canvas.Top="10" Canvas.Right="10">
           <Image Source="{Binding Path=Photo}"/>
        </Border>
        <Label Name="lblCopyright" Canvas.Bottom="10"
         Canvas.Right="10"
         Content="Micro Application 2006" />
      </Canvas>
   </DockPanel>
</Window>
```

Remarque

Autres modifications

Au passage, la gestion des événements `Click` a été ajoutée dans la barre d'outils pour faire appel aux méthodes `Click_Next` et `Click_Prev`.



▲ **Figure 7-7 :** Liaison à un DataSet

La fenêtre est affichée avec toutes les informations du premier enregistrement.

Astuce

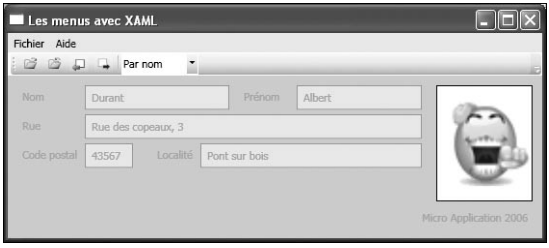
Contenu du champ Photo

Le champ Photo contient le chemin des images et non l'image elle-même.

En revanche, rien ne se passe si vous cliquez sur **Suivant**. Nous devons encore ajouter la gestion des changements d'enregistrement.

```
Public Sub Click_Prev(ByVal sender As Object _
                      , ByVal e As RoutedEventArgs)
    ' Code pour précedent
    If m_i > 0 Then
        m_i = m_i - 1
        Me.DataContext = m_data.Carnet_adresses.Rows(m_i)
    End If
End Sub

Public Sub Click_Next(ByVal sender As Object _
                      , ByVal e As RoutedEventArgs)
    ' Code pour suivant
    If m_i < m_data.Carnet_adresses.Rows.Count - 1 Then
        m_i = m_i + 1
        Me.DataContext = m_data.Carnet_adresses.Rows(m_i)
    End If
End Sub
```

Le champ `m_i` permet de conserver en mémoire la position courante dans la `DataTable`. Ce code intègre les instructions pour éviter de dépasser les limites de la table. Comme vous pouvez le constater, il suffit d'adapter le `DataContext` pour qu'il contienne la ligne voulue. Maintenant, l'enregistrement change lorsque vous demandez le suivant ou le précédent.



▲ **Figure 7-8 :** Changer d'enregistrement

Idéalement, il faudrait désactiver les boutons **Précédent** et **Suivant** des différents menus quand ceux-ci ne peuvent être réalisés.

Renvoi *Pour y arriver, reportez-vous au chapitre* ***Rendre un élément du menu inactif*** *page 172.*

Vous pouvez modifier une valeur dans l'écran, changer d'enregistrement et revenir sur celui que vous avez modifié, les modifications sont toujours présentes.

Attention

Enregistrement dans la base de données

Les données sont bien sauvées dans le DataSet mais le code en l'état n'enregistre pas le DataSet dans la base de données. Les modifications sont dès lors perdues au moment où vous quittez le programme.

Par exemple, nous pouvons enregistrer les données dans la base de données lors de l'opération de fermeture. Modifiez le code pour refléter le code ci-dessous.

```
Private m_adapter As _
 DBXAMLDataSetTableAdapters.Carnet_adressesTableAdapter

Public Sub New()
   InitializeComponent()
   m_adapter = New _
    DBXAMLDataSetTableAdapters.Carnet_adressesTableAdapter
   m_adapter.Fill(m_data.Carnet_adresses)
End Sub

Public Sub Click_Fermer(ByVal sender As Object _
                      , ByVal e As RoutedEventArgs)
   Me.mnuFermer.IsEnabled = False
   Me.mnuOuvrir.IsEnabled = True
   Me.mnuEdition.Visibility = _
                     Windows.Visibility.Collapsed
   Me.frmData.IsEnabled = False
   m_adapter.Update(m_data)
End Sub
```

Remarque

Nouveau champ dans la classe

L'objet m_adapter précédemment défini comme variable locale dans le constructeur devient un champ de la classe. De cette façon, il peut être utilisé dans la méthode Click_Fermer.

La base de données sera dès lors correctement enregistrée.

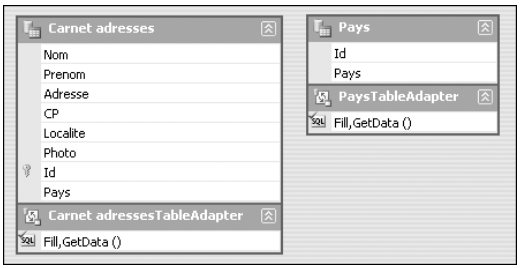
Astuce

Plusieurs sources de données

Dans l'exemple, nous avons travaillé avec un seul `DataContext` défini pour tout l'écran. Il est possible de définir un `DataContext` différent pour chaque élément de la fenêtre.

Il est également possible d'utiliser une table pour charger un contrôle de type liste.

Si vous souhaitez reproduire l'exemple, modifiez votre base de données et recréez le schéma XSD pour refléter le graphique ci-dessous.



◄ **Figure 7-9 :** Nouveau schéma XSD

Renvoi *Pour créer le schéma, reportez-vous à la page 192.*

Ensuite, nous devons apporter quelques modifications au code .NET.

```
Public Sub New()
   InitializeComponent()
   m_adapter = New _
    DBXAMLDataSetTableAdapters.Carnet_adressesTableAdapter
   m_adapter.Fill(m_data.Carnet_adresses)
   Dim adapter As New _
    DBXAMLDataSetTableAdapters.PaysTableAdapter
   adapter.Fill(m_data.Pays)
End Sub
```

Non seulement nous chargeons le membre `m_data` avec la `DataTable` *Carnet_adresses* mais nous chargeons également la table *Pays*.

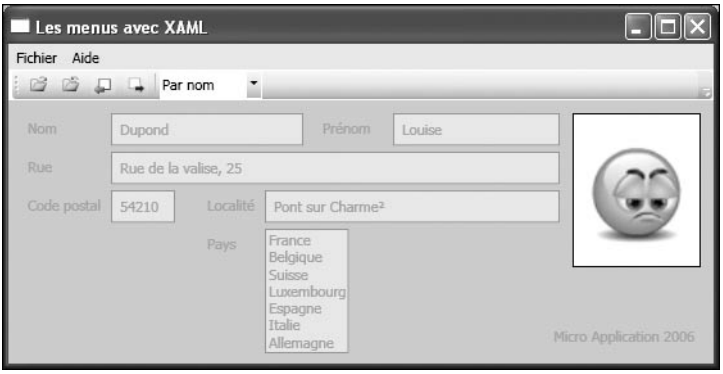
```
Private Sub Win_loaded(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
    m_i = 0
    Me.DataContext = m_data.Carnet_adresses.Rows(m_i)
    Me.lstPays.DataContext = m_data
End Sub
```

La table *Pays* est maintenant chargée dans le DataSet et le DataContext de la liste est différent de celui du reste de l'écran.

Ajoutez ces balises dans le code XAML.

```
<Label Canvas.Top="100" Canvas.Left="150">
   Pays
</Label>
<ListBox Name="lstPays" Canvas.Top="100"
 Canvas.Left="200"
 ItemsSource="{Binding Path=Pays}"
 DisplayMemberPath="Pays"/>
```

Remarquez que la source est non pas un champ mais une table. C'est l'attribut DisplayMemberPath qui précise le nom du champ à afficher. Dans cet exemple, le nom est le même car il s'agit du champ Pays dans la table *Pays*.



▲ **Figure 7-10 :** ListBox liée à un DataSet

Cette solution fonctionne parfaitement pour charger la liste mais, du coup, il ne nous est pas possible de lier la valeur à notre carnet d'adresses.

La solution la plus simple est de modifier à nouveau le code .NET.

```
Private Sub Win_loaded(ByVal sender As Object _
                       , ByVal e As RoutedEventArgs)
    m_i = 0
    Me.DataContext = m_data.Carnet_adresses.Rows(m_i)
    Dim bind As New Binding("Pays")
    bind.Source = m_data
    Me.lstPays.SetBinding(ListBox.ItemsSourceProperty _
                          , bind)
End Sub
```

Le code XAML devient alors :

```
<ListBox Name="lstPays" Canvas.Top="100"
 Canvas.Left="200"
 DisplayMemberPath="NomPays"
 SelectedValue="{Binding Path=Pays}"
 SelectedValuePath="Id"
 />
```

De cette façon, une source spécifique est définie pour la liste et la source par défaut pour la valeur.

Cet exemple démontre non seulement qu'il est possible de réaliser des liaisons vers divers objets mais également que la liaison peut être effectuée sur différents attributs. Vous pourriez par exemple définir une liaison sur la couleur du fond.

7.2 Lier les données à un objet métier

Au lieu de lier les données à un `DataSet`, vous pouvez également les lier à un objet quelconque. Généralement, il s'agira d'un objet dit métier. En effet, en programmation professionnelle, les développements sont généralement divisés en trois couches, la couche de liaison à base de données, la couche métier, dite business, et la couche de présentation. La couche de présentation lit les données non pas directement dans la couche d'accès à la base de données mais uniquement dans la couche métier, qui prendra en charge une éventuelle transformation des données.

Nous allons construire un objet métier capable de recevoir les données de notre exemple.

```
Public Class Business

    Private m_nom As String
    Private m_prenom As String
    Private m_adresse As String
    Private m_cp As String
    Private m_localite As String
    Private m_pays As Integer

    Public Property Nom() As String
        Get
            Return m_nom
        End Get
        Set(ByVal value As String)
            m_nom = value.ToUpper()
        End Set
```

```
    End Property

    Public Property Prenom() As String
        Get
            Return m_prenom
        End Get
        Set(ByVal value As String)
            m_prenom = valueli
        End Set
    End Property
```

Comme vous pouvez le voir, il s'agit essentiellement de définir des membres privés et les propriétés qui leur sont associées. Vous pourriez être tenté de définir directement les membres comme publics et ainsi de vous passer des propriétés. Toutefois, il vaut mieux suivre dès le départ les bonnes pratiques et respecter cette règle qui apportera dans l'évolution de votre classe beaucoup plus de souplesse. Sans compter que le membre défini peut pour des raisons métier ou techniques être stocké d'une certaine manière et présenté aux utilisateurs de la classe d'une autre façon.

```
    Public Property Adresse() As String
        Get
            Return m_adresse
        End Get
        Set(ByVal value As String)
            m_adresse = value
        End Set
    End Property

    Public Property CP() As String
        Get
            Return m_cp
        End Get
        Set(ByVal value As String)
            m_cp = value
        End Set
    End Property

    Public Property Localite() As String
        Get
            Return m_localite
        End Get
        Set(ByVal value As String)
            m_localite = value
        End Set
    End Property
```

```
    Public Property Pays() As Integer
        Get
            Return m_pays
        End Get
        Set(ByVal value As Integer)
            m_pays = value
        End Set
    End Property
```

Pour une question de simplicité, la méthode de chargement des données est incluse dans l'objet `Business`. Il existe beaucoup de techniques différentes pour réaliser les couches, mais nous entrons là dans des problèmes d'architecture bien éloignés de ce qui nous occupe. Toutefois, même avec XAML et peut-être encore plus avec XAML, pensez à l'architecture que vous allez implémenter dans votre programme avant de commencer la moindre lige de code.

```
    Public Sub Charger(ByVal val As Short)
        If val = 1 Then
            m_nom = "Dupond"
            m_prenom = "Louis"
            m_adresse = "Rue des coteaux, 23"
            m_localite = "Pillion"
            m_cp = "23456"
            m_pays = 22
        Else
            m_nom = "Durand"
            m_prenom = "Albert"
            m_adresse = "Rue des poteaux, 2"
            m_localite = "Paille"
            m_cp = "23765"
            m_pays = 24
        End If
    End Sub

End Class
```

Le but de cet exemple n'est pas de vous apprendre à travailler en couche. La classe métier qui vous est présentée ici est très simplifiée et la méthode de chargement des données n'est là que pour permettre d'afficher simplement un résultat. Typiquement, nous devrions trouver dans la classe des méthodes de manipulation, de contrôle, de transformation des données. L'interaction entre l'objet métier et la couche d'accès aux données n'est pas gérée par XAML. Il s'agit de .NET pur.

Le code .NET doit être adapté pour charger l'objet métier.

```
Partial Public Class Window1
    Inherits Window
```

En tout premier, nous devons déclarer notre objet métier comme membre de la classe. Notez que nous avons également conservé le DataSet. Il est encore nécessaire pour charger la liste des pays.

```
    Private m_data As New DBXAMLDataSet
    Private m_business As New Business
```

Dans le constructeur, nous chargeons les données dans le DataSet et dans l'objet métier.

```
    Public Sub New()
        InitializeComponent()
        Dim adapter As New
          DBXAMLDataSetTableAdapters.PaysTableAdapter
        adapter.Fill(m_data.Pays)
        m_business.Charger(1)
    End Sub
```

La liaison proprement dite est réalisée lors du chargement de la fenêtre.

```
    Private Sub Win_loaded(ByVal sender As Object
                         , ByVal e As RoutedEventArgs)
        Dim bind As New Binding("Pays")
        bind.Source = m_data
        Me.lstPays.SetBinding(ListBox.ItemsSourceProperty
                            , bind)
        Me.DataContext = m_business
    End Sub

    Public Sub Click_Prev(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        ' Code pour précedent
        m_business.Charger(1)
        Me.DataContext = Nothing
        Me.DataContext = m_business
    End Sub

    Public Sub Click_Next(ByVal sender As Object _
                    , ByVal e As RoutedEventArgs)
        ' Code pour suivant
        m_business.Charger(2)
        Me.DataContext = Nothing
        Me.DataContext = m_business
    End Sub

End Class
```

Comme c'était le cas pour le DataSet, l'objet métier est déclaré comme champ de la classe. Il est initialisé dans le constructeur et placé comme source de

données dans la méthode `Win_Load`. Il manque bien évidemment l'enregistrement des données de l'objet métier mais, là encore, il s'agit de programmation .NET classique.

Astuce

Rafraîchissement

Afin de provoquer le rafraîchissement de l'écran lorsque les valeurs sont rechargées, la valeur `Nothing` est chargée dans l'attribut `DataContext` et est ensuite à nouveau chargée par `m_business`.

Selon les circonstances, vous pourriez avoir plusieurs objets métier et changer le `DataContext` pour qu'il pointe sur l'un ou l'autre. Dans ce cas, typiquement, vous aurez une collection d'objets métier.

Remarque

Le fichier XAML n'est pas modifié

Les liaisons se font sur les noms des propriétés de la classe. Comme nous avons donné les mêmes noms à nos propriétés que les noms des champs dans le schéma *xsd*, le code XAML peut rester tel que.

En définitive, cet exemple démontre qu'il est non seulement possible de lier n'importe quelle propriété à une donnée externe mais également que la donnée externe peut elle-même être conservée dans n'importe quelle classe d'objet.

7.3 Lier les données sans utiliser le code .NET

Jusque-là, nous avons vu comment lier les contrôles à un objet mais UNIQUEMENT *via* le code .NET. Il est également possible de réaliser cette liaison directement dans le code XAML. Pour cela, vous disposez de deux outils différents. Le `XmlDataProvider` pour récupérer les données d'une source XML ou `ObjectDataProvider` pour récupérer les données d'un objet. Nous allons en premier voir comment récupérer les informations dans la source XML.

Tout d'abord, nous devons créer un fichier de données. Recopiez le code XML suivant dans un fichier que vous nommez *Data.xml*.

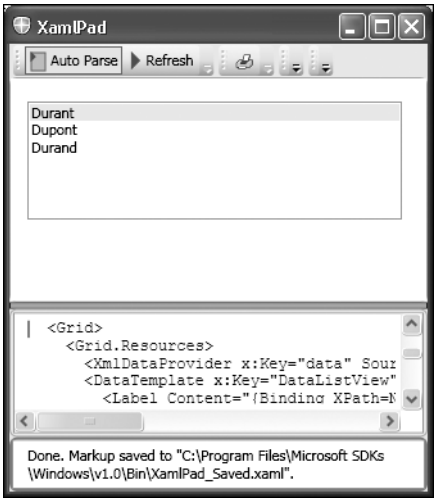
```
<Table>
   <Name>Direction</Name>
   <Records>
```

```
      <Record>
         <Nom>Durant</Nom>
         <Prenom>Louis</Prenom>
         <Titre>Directeur general</Titre>
      </Record>
      <Record>
         <Nom>Dupont</Nom>
         <Prenom>Leon</Prenom>
         <Titre>Product manager</Titre>
      </Record>
      <Record>
         <Nom>Durand</Nom>
         <Prenom>Carine</Prenom>
         <Titre>Directrice generale adjointe</Titre>
      </Record>
   </Records>
</Table>
```

Ce fichier contient donc une simulation de table dont le nom serait `Direction` et qui contient trois enregistrements. La structure de ce fichier XML n'est absolument pas une structure obligatoire et n'est reprise qu'à titre d'exemple.

Une fois ce fichier créé, nous pouvons maintenant réaliser notre page XAML qui va lire directement ce fichier. Les données seront affichées dans une `ListView`. Ce sera pour nous l'occasion de découvrir ce contrôle, qui se prête particulièrement bien à l'affichage de données enregistrées dans une source externe.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="data" Source=" Data.XML"/>
      </Grid.Resources>
      <Grid.DataContext>
         <Binding Source="{StaticResource data}"
          XPath="/Table/Records"/>
      </Grid.DataContext>
      <ListView Height="92" Margin="13,18,19,0"
       Name="MaListView" VerticalAlignment="Top"
       ItemsSource="{Binding XPath=Record/Nom}"
   </Grid>
</Page>
```



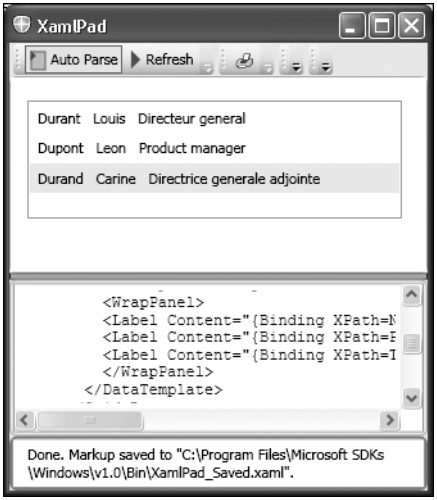
◄ **Figure 7-11 :** Affichage du contenu d'un fichier XML dans une ListView

Comme vous pouvez le constater, la balise `XmlDataProvider` est extrêmement simple d'utilisation puisqu'il ne s'agit que de donner un nom *via* l'attribut `x:Key` et de définir le fichier en utilisant l'attribut `Source`. Comme pour les techniques précédentes, nous devons utiliser le `DataContext`. Cette fois, nous le faisons directement dans le fichier XAML puisque la source est définie en tant que ressource statique. L'attribut `XPath` va permettre de déterminer dans le fichier XML le nœud qui contient les données qui nous intéressent. Il ne reste plus alors qu'à définir la `ListView`. En dehors de l'attribut `ItemSource`, qui permet de définir les données associées, rien de bien neuf. Notez que, pour définir les données, nous utilisons à nouveau `XPath`. Le contenu du nœud *Nom* inscrit dans chaque nœud *Record* de la source de données (`DataContext`) servira à définir un élément de la liste.

Si nous désirons afficher plus d'une information par liste, vous pouvez utiliser l'attribut `ItemTemplate` de notre `ListView`. Pour cela, nous devons au préalable définir un `DataTemplate`. Le `DataTemplate` sert à décrire le contenu et la façon de représenter chacun des éléments de la liste. Il se place également dans la zone des ressources.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="data" Source=" Data.XML"/>
```

```
        <DataTemplate x:Key="DataListView">
           <WrapPanel>
              <Label Content="{Binding XPath=Nom}"/>
              <Label Content="{Binding XPath=Prenom}"/>
              <Label Content="{Binding XPath=Titre}"/>
           </WrapPanel>
        </DataTemplate>
     </Grid.Resources>
     <Grid.DataContext>
        <Binding Source="{StaticResource data}"
         XPath="/Table/Records"/>
     </Grid.DataContext>
     <ListView Height="92" Margin="13,18,19,0"
      Name="MaListView" VerticalAlignment="Top"
      ItemsSource="{Binding XPath=Record}"
      ItemTemplate="{StaticResource DataListView}"/>
  </Grid>
</Page>
```



◄ **Figure 7-12 :** Affichage d'une ListView en utilisant un DataTemplate

Remarque

Contenu d'un DataTemplate

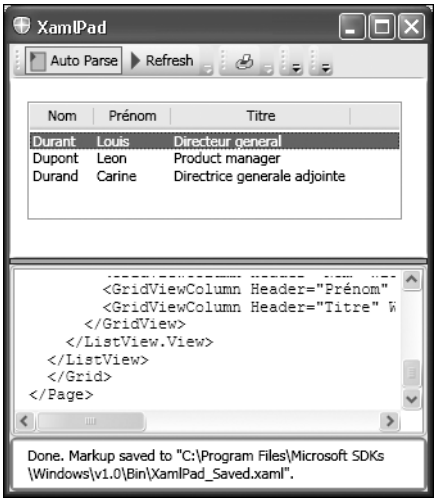
Comme vous pouvez le remarquer, un `DataTemplate` peut contenir n'importe quel code XAML ou presque.

Un autre moyen d'afficher plus d'informations est d'utiliser un GridView. Le GridView est non pas à proprement parler un autre contrôle mais plutôt une extension qui se place dans la propriété View de ListView. Il permet d'afficher les informations dans des colonnes séparées et même de donner un titre à ces colonnes. Si nous reprenons le même exemple, nous devrons le modifier comme suit.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="data" Source=" Data.XML"/>
      </Grid.Resources>
      <Grid.DataContext>
         <Binding Source="{StaticResource data}"
          XPath="/Table/Records"/>
      </Grid.DataContext>
      <ListView Height="92" Margin="13,18,19,0"
       Name="MaListView" VerticalAlignment="Top"
       ItemsSource="{Binding XPath=Record}">
         <ListView.View>
```

Dans cet exemple, le GridView contient trois colonnes, une pour le nom, une pour le prénom et une pour le titre.

```
            <GridView AllowsColumnReorder="True">
               <GridViewColumn Header="Nom"
                Width="50"
                DisplayMemberBinding=
                     "{Binding XPath=Nom}"/>
               <GridViewColumn Header="Prénom"
                Width="60"
                DisplayMemberBinding=
                     "{Binding XPath=Prenom}"/>
               <GridViewColumn Header="Titre"
                Width="140"
                DisplayMemberBinding=
                     "{Binding XPath=Titre}"/>
            </GridView>
         </ListView.View>
      </ListView>
   </Grid>
</Page>
```



◄ **Figure 7-13 :** Affichage d'un GridView

Astuce

Déplacer les colonnes

Pour permettre à l'utilisateur de déplacer les colonnes par glisser-déposer, vous devez assigner la valeur `True` à l'attribut `AllowsColulmnReorder` dans la balise `GridView`.

Si vous préférez donner un aspect plus structuré à votre affichage, vous utiliserez probablement un `TreeView`. Sur la base du même exemple, nous pouvons construire un petit `TreeView` mais, afin de lui donner un aspect plus hiérarchisé, il y a lieu de modifier au préalable notre source de données.

```
<Table>
   <Name>
      Direction
   </Name>
   <Records>
      <Administration>
         <Record>
            <Nom>Durant</Nom>
            <Prenom>Louis</Prenom>
            <Titre>Directeur general</Titre>
         </Record>
         <Record>
            <Nom>Durand</Nom>
            <Prenom>Carine</Prenom>
```

```
            <Titre>Directrice generale adjointe</Titre>
         </Record>
      </Administration>
      <Production>
         <Record>
            <Nom>Dupont</Nom>
            <Prenom>Leon</Prenom>
            <Titre>Product manager</Titre>
         </Record>
      </Production>
   </Records>
</Table>
```

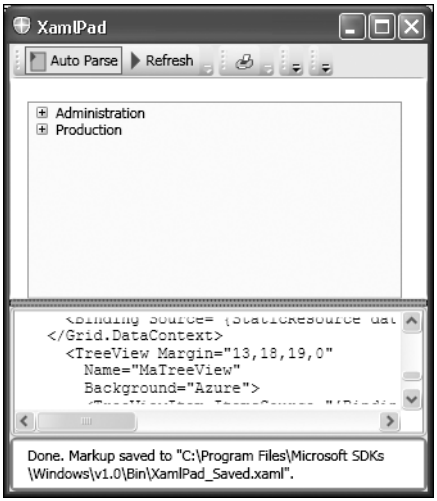
Avec les nœuds *Administration* et *Production*, les membres de la direction sont maintenant groupés selon leurs tâches. Nous pouvons représenter cette structure dans le `TreeView`.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="data"
          Source="Data2.XML"/>
         <DataTemplate x:Key="Admin">
            <WrapPanel>
               <Label Foreground="Blue"
                Content="{Binding XPath=Nom}"/>
               <Label Content="{Binding XPath=Prenom}"/>
               <Label Content="{Binding XPath=Titre}"/>
            </WrapPanel>
         </DataTemplate>
```

Nous pouvons définir un deuxième `DataTemplate` sur la même source. Dans l'exemple, seule la couleur est modifiée, mais vous pourriez changer tous les éléments.

```
         <DataTemplate x:Key="Prod">
            <WrapPanel>
               <Label Foreground="Green"
                Content="{Binding XPath=Nom}"/>
               <Label Content="{Binding XPath=Prenom}"/>
               <Label Content="{Binding XPath=Titre}"/>
            </WrapPanel>
         </DataTemplate>
      </Grid.Resources>
      <Grid.DataContext>
```

```
        <Binding Source="{StaticResource data}"
         XPath="/Table/Records"/>
      </Grid.DataContext>
      <TreeView Margin="13,18,19,0"
       Name="MaTreeView"
       Background="Azure">
        <TreeViewItem
         ItemsSource="{Binding XPath=Administration/Record}"
         Header="Administration"
         ItemTemplate="{StaticResource Admin}"/>
        <TreeViewItem
         ItemsSource="{Binding XPath=Production/Record}"
         Header="Production"
         ItemTemplate="{StaticResource Prod}"/>
      </TreeView>
   </Grid>
</Page>
```

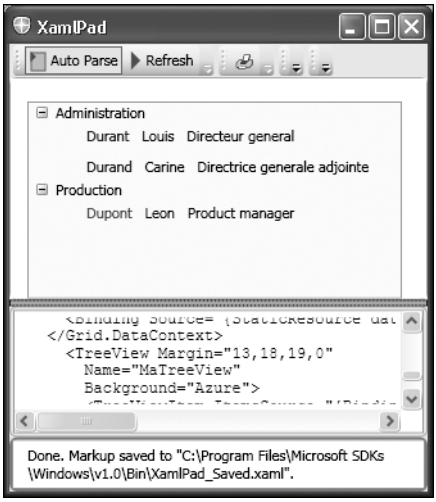


◄ **Figure 7-14 :** Affichage d'un TreeView

Comme vous pouvez le constater, chaque élément du `TreeView` possède sa propre source ou plutôt son propre chemin vers les données. Il peut aussi avoir sa propre représentation mais, généralement, le modèle (`DataTemplate`) sera le même pour l'ensemble des éléments.

Voyons maintenant comment réaliser une liaison avec un objet en utilisant `ObjectDataProvider`.

Nous devons tout d'abord créer une classe. Prenons comme exemple la classe `Auteur`, qui contient son nom et une collection de livres qu'il a écrits.



◄ **Figure 7-15 :** Le TreeView ouvert

Pour cela, il nous faut une classe `Livre`.

```
Public Class Livre
    Private _name As String
    Public ReadOnly Property Name() As String
        Get
            Return _name
        End Get
    End Property
```

Le nom du livre est initialisé dans le constructeur.

```
    Public Sub New(ByVal name As String)
        _name = name
    End Sub
End Class
```

Ensuite, nous avons besoin d'une collection typée pour stocker les objets `Livre`.

```
Public Class Livres
    Inherits ObservableCollection(Of Livre)

    Public Sub New()

    End Sub
End Class
```

Nous pouvons maintenant créer la classe Auteur.

```
Public Class Auteur
    Private _name As String
    Private _livres As Livres

    Public ReadOnly Property Name() As String
        Get
            Return _name
        End Get
    End Property

    Public ReadOnly Property Livres() As Livres
        Get
            Return _livres
        End Get
    End Property
```

L'auteur et les livres sont initialisés dans le constructeur.

```
    Public Sub New()
        _name = "Stephen King"
        _livres = New Livres
        _livres.Add(New Livre("La tour sombre"))
        _livres.Add(New Livre("Ca"))
        _livres.Add(New Livre("La peau sur les os"))
    End Sub
End Class
```

Maintenant, nous pouvons passer au code XAML et à l'utilisation d'ObjectDataProvider. La propriété ObjectType va nous permettre de spécifier le type d'objet qui sera créé.

Attention

Objet créé dans le code .NET

ObjectDataProvider ne fait pas la liaison avec un objet existant mais instancie un nouvel objet.

Pour réaliser la liaison avec un objet défini dans le code .NET, vous devez utiliser la méthode vue précédemment.

Le paramètre ConstructorParameters, que nous n'utiliserons pas ici, permet de transmettre des paramètres au constructeur. Les autres techniques utilisées ont déjà été vues précédemment.

```
<Window xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:src="clr-namespace:ObjectProviderExemple"
>
   <Window.Resources>
      <ObjectDataProvider x:Key="sk"
       ObjectType="{x:Type src:Auteur}"/>
      <DataTemplate x:Key="NomAuteur">
         <TextBlock Text="{Binding Path=Name}" />
      </DataTemplate>
   </Window.Resources>
   <StackPanel DataContext=
    "{Binding Source={StaticResource sk}}"
    Margin="10,10,10,0">
      <TextBlock Text="{Binding Path=Name}"
       Background="LightGray" />
      <ListView Background="LemonChiffon"
       ItemsSource="{Binding Path=Livres}"
       ItemTemplate="{DynamicResource NomAuteur}"
      />
   </StackPanel>
</Window>
```

◄ **Figure 7-16 :** Binding avec ObjectDataProvider

Remarque

ObjectProviderExemple

`ObjectProviderExemple` est le nom du projet et donc par défaut celui de l'assembly.

Les paramètres `ObjectInstance`, `MethodName` et `MethodParameters` étendent encore les possibilités de cette classe en permettant entre autres l'accès à des méthodes de l'objet créé.

7.4 Checklist

Dans ce chapitre sur la connexion aux données, nous avons essentiellement vu :

- comment lier un objet `DataSet` ou autre avec un contrôle en utilisant les propriétés `DataContext` et `Source` ;
- comment lier les données en utilisant `Binding` ;
- comment lier les données à un fichier XML en utilisant `XmlDataProvider` ;
- comment afficher les données dans une `ListView` ou de manière plus élaborée avec un `GridView` ;
- comment afficher les données dans une arborescence avec `TreeView` ;
- comment lier les données à un objet en utilisant `ObjectDataProvider`.

Fonctionnalités avancées

Appliquer des transformations sur les contrôles 220
Créer une ressource 223
Créer un style 227
Checklist 247

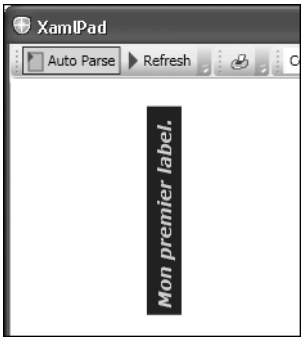
8.1 Appliquer des transformations sur les contrôles

XAML nous offre quelques possibilités pour manipuler la disposition des contrôles. Nous n'entrerons pas dans les détails. Toutefois, au travers des quelques exemples présentés dans ce chapitre et si vous avez un jour besoin d'une telle fonctionnalité, vous devriez pouvoir facilement trouver les paramètres qui vous conviennent.

Comme premier exemple, nous allons effectuer une rotation sur une étiquette.

Pour l'exemple, reprenons l'étiquette définie page 28.

```
<Label Name="lblPremierLabel"
 HorizontalAlignment="Center" VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14" FontWeight="Heavy"
 FontStyle="Italic" FontStretch="UltraExpanded"
 Foreground="Aquamarine" Background="Blue">
   Mon premier label.
   <Label.RenderTransform>
      <RotateTransform Angle="270" />
   </Label.RenderTransform>
</Label>
```



◄ **Figure 8-1 :** Une étiquette verticale

Accéder à une propriété grâce à un nœud fils

Pour la première fois, nous utilisons une propriété de la classe non pas en la définissant grâce à un attribut mais bien comme un nœud fils de notre nœud `Label`.

marque

Il est également possible de réaliser une modification d'échelle.

```
<Label Name="lblPremierLabel"
 HorizontalAlignment="Center" VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14" FontWeight="Heavy"
 FontStyle="Italic" FontStretch="UltraExpanded"
 Foreground="Aquamarine" Background="Blue">
   Mon premier label.
   <Label.RenderTransform>
      <ScaleTransform CenterY="20" ScaleX="1" ScaleY="7" />
   </Label.RenderTransform>
</Label>
```

Notez la présence de l'attribut `CenterY`, qui comme `CenterX` permet de modifier la position du contrôle en même temps que lui est appliquée la transformation.

◄ **Figure 8-2 :** Une étiquette étirée

Nous pouvons aussi adapter l'oblicité sur deux axes.

```
<Label Name="lblPremierLabel"
 HorizontalAlignment="Center" VerticalAlignment="Center"
 FontFamily="Verdana" FontSize="14" FontWeight="Heavy"
 FontStyle="Italic" FontStretch="UltraExpanded"
 Foreground="Aquamarine" Background="Blue">
   Mon premier label.
   <Label.RenderTransform>
      <SkewTransform AngleX="45" AngleY="20" />
   </Label.RenderTransform>
</Label>
```

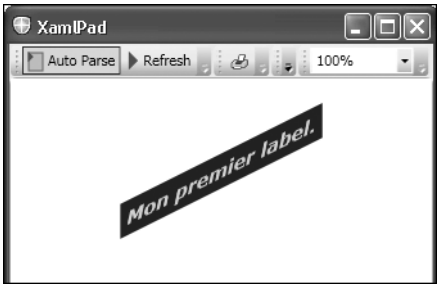
◄ **Figure 8-3 :** Une étiquette en 3D

Si vous ne trouvez pas votre bonheur dans ces transformations, vous pouvez toujours utiliser une transformation grâce à une matrice.

```
<Label Name="lblPremierLabel"
  HorizontalAlignment="Center" VerticalAlignment="Center"
  FontFamily="Verdana" FontSize="14" FontWeight="Heavy"
  FontStyle="Italic" FontStretch="UltraExpanded"
  Foreground="Aquamarine" Background="Blue"
  RenderTransform= "1,-0.5,0,1,0,0">
   Mon premier label.
</Label>
```

Nous revenons avec cette commande dans la structure plus traditionnelle d'un attribut de notre balise.

Chaque chiffre de la matrice applique une transformation à notre label.



◄ **Figure 8-4 :** Une étiquette oblique

À vous de découvrir le résultat obtenu en modifiant chacun des paramètres.

Les transformations appliquées précédemment sur un `Label` peuvent parfaitement être utilisées avec d'autres contrôles. Par exemple, l'effet sur un texte long est, pour qui a l'habitude des limites de l'API Win32, toujours très surprenant.

```
<TextBlock Name="blckTexte"
 MaxWidth="200" MaxHeight="70"
 TextWrapping="Wrap" TextTrimming="WordEllipsis"
 Margin="5,5,5,5" TextAlignment="Justify" >
   Nous sommes maintenant arrivés à notre deuxième contrôle.
   Comme vous avez déjà pu le constater, XAML est à la fois
   simple d'utilisation et performant.
   <TextBlock.RenderTransform>
      <RotateTransform CenterX="100" CenterY="100"
        Angle="270"/>
   </TextBlock.RenderTransform>
</TextBlock>
```

◄ **Figure 8-5** : Un bloc de texte vertical

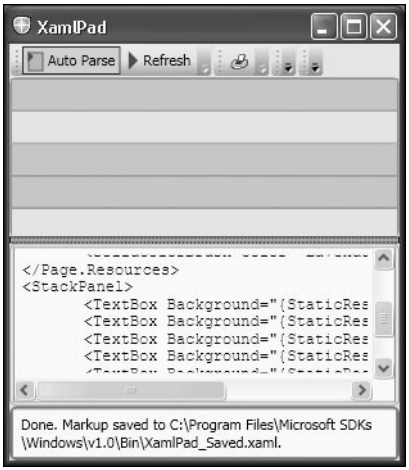
8.2 Créer une ressource

Une ressource en XAML est un élément que vous codez afin qu'il soit réutilisé au sein du conteneur dans lequel elle est définie. L'utilité de la ressource est de

pouvoir réutiliser le code sans devoir le réécrire complètement. Ce qui permet un gain de temps mais aussi limite les risques de bug ou d'oubli en cas de modification.

Prenons un exemple simple de ressource.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Page.Resources>
      <SolidColorBrush Color="Gold" x:Key="Background1"/>
      <SolidColorBrush Color="Lavender" x:Key="Background2"/>
   </Page.Resources>
   <StackPanel>
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background2}"/>
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background2}"/>
   </StackPanel>
</Page>
```

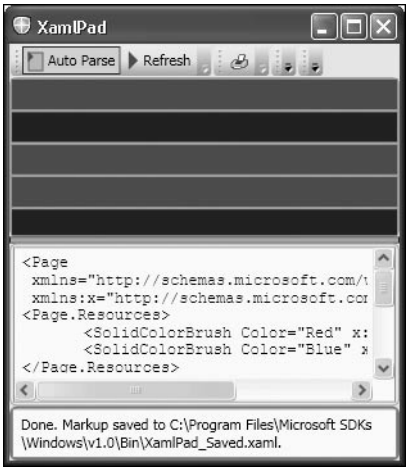


◄ **Figure 8-6 :** Utiliser des ressources

Le nom de la ressource est défini grâce à l'attribut `x :key`. La ressource est accédée en assignat `{StaticResource NomDeLaRessource}` à l'attribut cible.

Si vous changez la ressource, toute la page est directement affectée. Il n'est pas nécessaire de modifier chaque contrôle.

```
<Page.Resources>
   <SolidColorBrush Color="Red" x:Key="Background1"/>
   <SolidColorBrush Color="Blue" x:Key="Background2"/>
</Page.Resources>
```



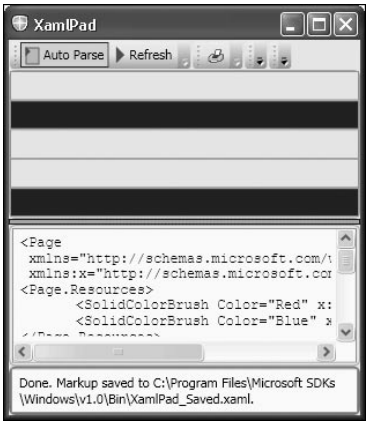
◄ **Figure 8-7 :** Ressources modifiées

Nous avons utilisé les ressources de manière statique, il est toutefois possible de les utiliser de manière dynamique en remplaçant le mot clé `StaticResource` par `DynamicResource`. Cette possibilité n'est intéressante que quand la ressource peut être modifiée. Elle pourra alors être rechargée.

Il est possible d'avoir deux ressources portant le même nom pour autant qu'elles ne soient pas définies dans la même balise. Pour retrouver une ressource, XAML remonte niveau après niveau à la recherche de cette ressource. Pour la lisibilité du code, il est malgré tout fortement conseillé d'utiliser des noms différents.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Page.Resources>
     <SolidColorBrush Color="Red" x:Key="Background1"/>
      <SolidColorBrush Color="Blue" x:Key="Background2"/>
   </Page.Resources>
   <StackPanel>
      <StackPanel.Resources>
         <SolidColorBrush Color="Lavender"
          x:Key="Background1"/>
      </StackPanel.Resources>
```

```
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background2}"/>
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background1}"/>
      <TextBox Background="{StaticResource Background2}"/>
   </StackPanel>
</Page>
```



◄ **Figure 8-8 :** Ressources de même nom

Les ressources sont aussi un moyen simple pour partager un menu contextuel entre plusieurs éléments.

```
<Window.Resources>
   <ContextMenu x:Key="CtxtMnu">
      <MenuItem Header="Copier"/>
      <MenuItem Header="Coller"/>
   </ContextMenu>
</Window.Resources>
…
<TextBox Name="txtNom"  Canvas.Top="10"
 Canvas.Left="80" Width="150" MaxLength="30"
 CharacterCasing="Upper"
 Text ="{Binding Path=Nom}"
 ContextMenu="{StaticResource CtxtMnu}" />
<Label Name="lblPrenom" Canvas.Top="10"
 Canvas.Left="240">
   Prénom
</Label>
<TextBox Name="txtPrenom"  Canvas.Top="10"
 Canvas.Left="300" Width="130" MaxLength="30"
 Text ="{Binding Path=Prenom}"
 ContextMenu="{StaticResource CtxtMnu}"/>
…
```

8.3 Créer un style

Les styles vont permettre de modifier l'apparence des contrôles, et cela sans répéter les modifications pour chaque contrôle individuellement.

Prenons un exemple simple. La couleur du fond des boîtes de texte ne vous convient pas. Bien sûr, vous pouvez changer cette couleur avec l'attribut `Background` sur chaque `TextBox` mais aussi utiliser un style.

Un style est défini comme ressource d'un conteneur. Nous choisirons de le placer comme ressource de la fenêtre.

```
<Window.Resources>
   <Style x:Key="MyTextBox">
      <Setter Property="TextBox.Background" Value="Gold"/>
   </Style>
</Window.Resources>
```

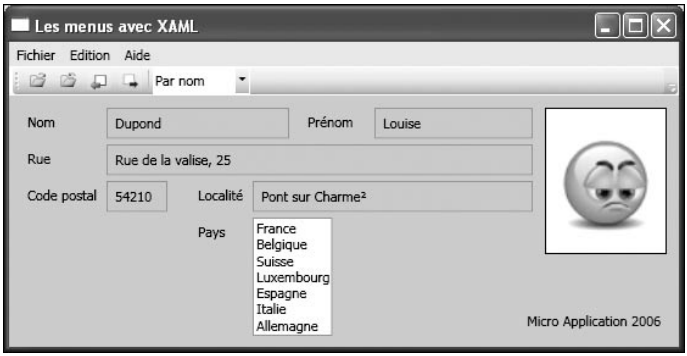
Si nous analysons ce code, la première chose à faire est de donner un nom à ce style avec l'attribut `x:Key`. La balise `Setter` va nous permettre de définir la propriété affectée.

Il ne nous reste qu'à demander l'utilisation du style. Cela se fait pour chaque contrôle car vous pourriez avoir plusieurs styles différents pouvant être appliqués au même type de contrôle.

Pour appliquer un style, il suffit d''utiliser l'attribut `Style`.

```
<TextBox Name="txtNom"  Canvas.Top="10"
 Canvas.Left="80" Width="150" MaxLength="30"
 CharacterCasing="Upper" Text ="{Binding Path=Nom}"
 Style="{StaticResource MyTextBox}">
```

Faites de même avec les autres boîtes de texte.



▲ **Figure 8-9 :** Mon premier style

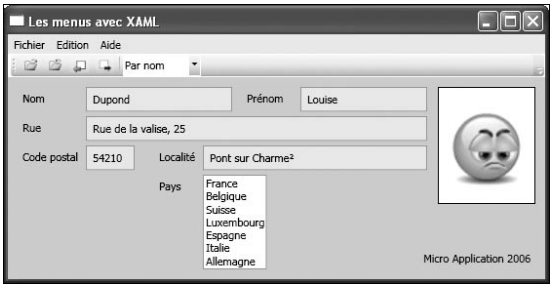
Attention

Affichage du style

Le style tel que nous l'avons défini n'affecte le contrôle que lorsqu'il est activé. Son apparence désactivée n'a pas changé. Pour voir les couleurs, vous devez donc utiliser l'option *Ouvrir* du menu.

Bien sûr, vous pouvez rétorquer qu'il a fallu modifier toutes les `TextBox` et que, dès lors, il aurait été aussi simple de changer l'attribut correspondant. Toutefois, si vous désirez maintenant changer la couleur, il suffit de la changer à un endroit. Changeons la couleur.

```
<Style x:Key="MyTextBox">
   <Setter Property="TextBox.Background" Value="Lavender"/>
</Style>
```



▲ **Figure 8-10 :** Changement rapide

Toutes les boîtes de texte concernées sont automatiquement adaptées.

Si vous trouvez toujours qu'il est fastidieux de déclarer le style pour chaque contrôle, il est également possible d'appliquer automatiquement un style à tous les contrôles d'un même type sans devoir le préciser pour chacun d'eux.

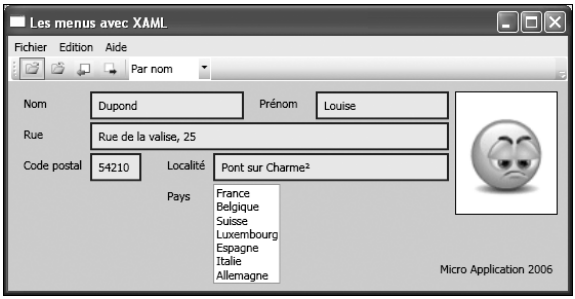
```
<Style TargetType="{x:Type TextBox}">
   <Setter Property="TextBox.Background" Value="Lavender"/>
</Style>
```

L'attribut `TargetType` permet de définir le type de contrôle qui doit être affecté.

Une autre bonne raison d'utiliser un style est qu'il peut regrouper non pas une mais un ensemble de modifications à apporter à la présentation. Et, dans ce cas, il sera beaucoup plus fastidieux de les réaliser pour chaque contrôle concerné.

```
<Style x:Key="MyTextBox">
   <Setter Property="TextBox.Background" Value="Lavender"/>
```

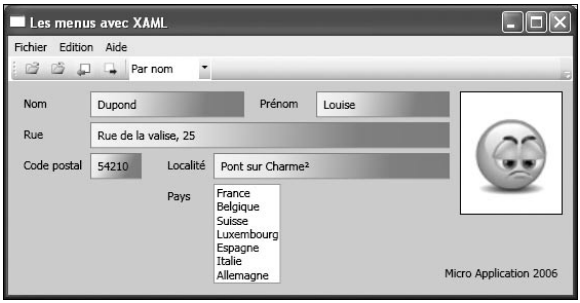
```
   <Setter Property="TextBox.BorderBrush" Value="Blue"/>
   <Setter Property="TextBox.BorderThickness" Value="2"/>
</Style>
```



▲ **Figure 8-11** : Changements multiples

Vous pouvez aussi avoir des changements complexes.

```
<Style x:Key="MyTextBox">
   <Setter Property="TextBox.Background">
      <Setter.Value>
         <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <LinearGradientBrush.GradientStops>
               <GradientStop Color="LavenderBlush" Offset="0" />
               <GradientStop Color="Lavender" Offset="0.25" />
               <GradientStop Color="Gray" Offset="0.75" />
               <GradientStop Color="SlateGray" Offset="1" />
            </LinearGradientBrush.GradientStops>
         </LinearGradientBrush>
      </Setter.Value>
   </Setter>
   <Setter Property="TextBox.BorderBrush" Value="Gray"/>
   <Setter Property="TextBox.BorderThickness" Value="1"/>
</Style>
```



▲ **Figure 8-12** : Style complexe

Comme vous pouvez le constater, il est possible de pousser très loin les modifications de présentation.

Une fois encore, nous pouvons voir qu'on peut accéder à une propriété comme à un attribut.

```
<Setter Property="TextBox.BorderBrush" Value="Gray"/>
```

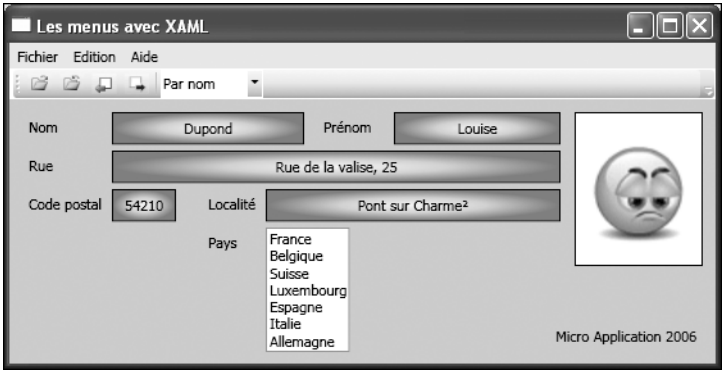
Ou comme à un nœud fils.

```
<Setter Property="TextBox.Background">
   <Setter.Value>
   </Setter.Value>
</Setter>
```

Une petite explication s'impose sur la classe `LinearGradientBrush`, que nous n'avons pas encore utilisée. Comme vous avez pu le constater, elle permet de réaliser un fondu linéaire entre plusieurs couleurs. Le fondu se fait sur la diagonale entre le coin supérieur gauche (coordonnées 0,0) et le coin inférieur droit (coordonnées 1,1). Les attributs `SartPoint` et `EndPoint` définissent le début et la fin de la zone où doit s'appliquer le fondu. La collection `GradientStops` permet de définir les couleurs et leurs positions sur la diagonale.

Vous pouvez également préférer un fondu radial en utilisant la classe `Radial-GradientBrush`..

```
<Style x:Key="MyTextBox">
   <Setter Property="TextBox.Background">
      <Setter.Value>
         <RadialGradientBrush>
            <RadialGradientBrush.GradientStops>
               <GradientStop Color="LavenderBlush"
                Offset="0" />
               <GradientStop Color="Lavender"
                Offset="0.2" />
               <GradientStop Color="LightGray"
                Offset="0.5" />
               <GradientStop Color="Gray" Offset="1" />
            </RadialGradientBrush.GradientStops>
         </RadialGradientBrush>
      </Setter.Value>
   </Setter>
   <Setter Property="TextBox.BorderBrush" Value="Black"/>
   <Setter Property="TextBox.BorderThickness" Value="1"/>
   <Setter Property="TextBox.TextAlignment"
    Value="Center"/>
</Style>
```



▲ **Figure 8-13 :** Fondu radial

Un autre avantage des styles est qu'il est possible d'hériter d'un autre style.

```
<Window.Resources>
   <Style x:Key="MyControl">
      <Setter Property="Control.Background">
         <Setter.Value>
            <RadialGradientBrush>
               <RadialGradientBrush.GradientStops>
                  <GradientStop Color="LavenderBlush"
                   Offset="0" />
                  <GradientStop Color="Lavender"
                   Offset="0.2" />
                  <GradientStop Color="LightGray"
                   Offset="0.5" />
                  <GradientStop Color="Gray" Offset="1" />
               </RadialGradientBrush.GradientStops>
            </RadialGradientBrush>
         </Setter.Value>
      </Setter>
      <Setter Property="Control.BorderBrush" Value="Black"/>
      <Setter Property="Control.BorderThickness" Value="1"/>
   </Style>
   <Style TargetType="{x:Type TextBox}" x:Key="MyTextBox"
    BasedOn="{StaticResource MyControl}">
      <Setter Property="TextBox.TextAlignment"
       Value="Center"/>
   </Style>
   <Style TargetType="{x:Type ListBox}" x:Key="MyListBox"
    BasedOn="{StaticResource MyControl}"/>
</Window.Resources>
```

▲ **Figure 8-14** : Héritage de style

Dans cet exemple, le style que nous avions défini est maintenant applicable au type `Control`. Le style peut être hérité par les autres styles destinés à d'autres contrôles. La propriété `TextAlignment` n'est modifiée que pour le style `MyTextBox`.

Astuce

Redéfinition d'un élément d'un style

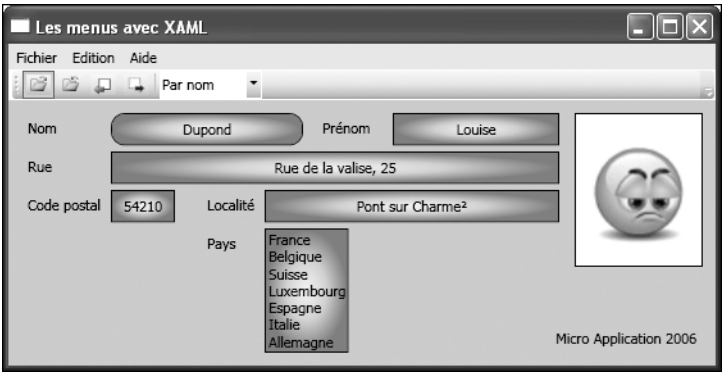
Si votre style hérite d'un autre, vous pouvez sans problème redéfinir des attributs déjà définis.

Un autre attribut important et souvent utilisé en conjonction avec les styles est l'attribut `Template`.

Si vous ne souhaitez changer l'attribut `Template` que pour un seul contrôle de votre écran, il est inutile de créer un style. Vous pouvez modifier cette propriété directement dans la balise du contrôle. Nous allons commencer en modifiant l'attribut `Template` du nom et nous créerons ensuite un style sur cette base. De cette façon, nous aurons utilisé les deux possibilités.

```
<TextBox Name="txtNom"  Canvas.Top="10" Canvas.Left="80"
 Width="150" MaxLength="30" CharacterCasing="Upper"
 Text ="{Binding Path=Nom}"
 >
   <TextBox.Template>

      <ControlTemplate>
         <Border CornerRadius="10" BorderThickness="1"
          BorderBrush="Black">
            <Border.Background>
```

```
            <RadialGradientBrush>
               <RadialGradientBrush.GradientStops>
                  <GradientStop Color="LavenderBlush"
                   Offset="0" />
                  <GradientStop Color="Lavender"
                   Offset="0.2" />
                  <GradientStop Color="LightGray"
                   Offset="0.5" />
                  <GradientStop Color="Gray"
                   Offset="1" />
               </RadialGradientBrush.GradientStops>
            </RadialGradientBrush>
         </Border.Background>
         <ScrollViewer x:Name="PART_ContentHost"
          Focusable="False"
          HorizontalAlignment="Center"/>
        </Border>
     </ControlTemplate>

  </TextBox.Template>

  <TextBox.ContextMenu>
     <ContextMenu>
        <MenuItem Header="Copier"/>
        <MenuItem Header="Coller"/>
     </ContextMenu>
  </TextBox.ContextMenu>
</TextBox>
```

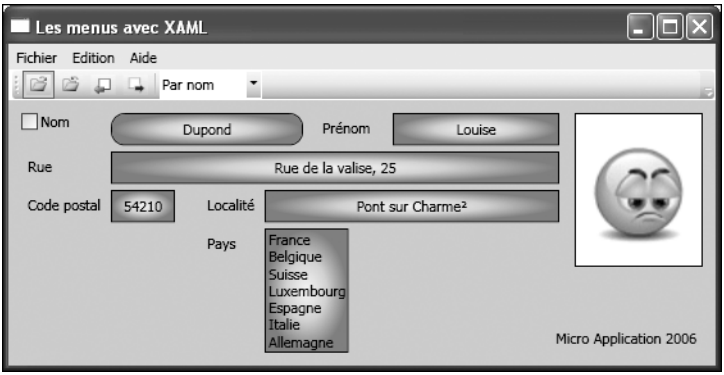


▲ **Figure 8-15** : Modification de la représentation d'une TextBox

Dans l'exemple, nous avons remplacé la représentation normale par une bordure arrondie avec les mêmes effets de fondu que le contrôle original, mais les coins sont arrondis. La valeur est contenue dans un `ScrollView`. Ceci est rendu possible grâce à la présence de l'attribut `x:Name="PART_ContentHost"`.

En utilisant cet attribut, il est possible de faire quasiment tout mais aussi n'importe quoi.

```
<Label Name="lblNom" Canvas.Top="10"
 Canvas.Left="10">
   Nom
   <Label.Template>
      <ControlTemplate>
         <CheckBox>
            <ContentPresenter
             x:Name="ContentSite" />
         </CheckBox>
      </ControlTemplate>
   </Label.Template>
</Label>
```



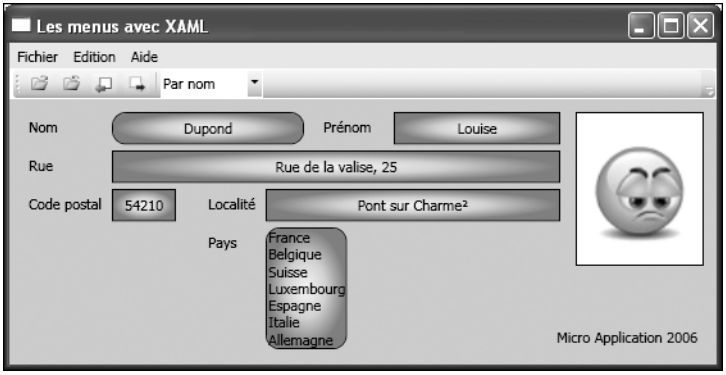
▲ **Figure 8-16 :** Représenter un label par une case à cocher

Dans l'exemple ci-dessus, la représentation du contrôle `Label` est remplacée par celle du contrôle `CheckBox`. Toutefois, il s'agit toujours d'un objet `Label`. Les propriétés de `CheckBox` comme `IsChecked` ne sont pas accessibles. Ce genre de remplacement n'a aucun intérêt mais démontre bien la dissociation faite entre l'objet et sa représentation.

Ce qu'il est possible de réaliser sur un contrôle à contenu simple, il est également possible de le réaliser pour un contrôle gérant une liste comme la `ListBox` utilisée dans l'exemple.

```
<ListBox Name="lstPays" Canvas.Top="100"
 Canvas.Left="200"
 DisplayMemberPath="Pays"
 SelectedValue="{Binding Path=Pays}"
 SelectedValuePath="Id">
   <ListBox.Template>
```

```
      <ControlTemplate>
         <Border CornerRadius="10" BorderThickness="1"
          BorderBrush="Black" >
            <Border.Background>
               <RadialGradientBrush>
                  <RadialGradientBrush.GradientStops>
                     <GradientStop Color="LavenderBlush"
                      Offset="0" />
                     <GradientStop Color="Lavender"
                      Offset="0.2" />
                     <GradientStop Color="LightGray"
                      Offset="0.5" />
                     <GradientStop Color="Gray" Offset="1" />
                  </RadialGradientBrush.GradientStops>
               </RadialGradientBrush>
            </Border.Background>
          <ScrollViewer
               Focusable="False"
               HorizontalAlignment="Center">
              <ItemsPresenter/>
            </ScrollViewer>
          </Border>
      </ControlTemplate>
   </ListBox.Template>
</ListBox>
```



▲ **Figure 8-17 :** Adapter la ListBox pour respecter le style des TextBox

Maintenant que nous avons vu comment modifier la représentation d'un contrôle, voyons comment inclure cela dans un style.

```
<Window.Resources>
  <Style x:Key="MyTextBox">
    <Setter Property="Template">
```

```
      <Setter.Value>
        <ControlTemplate>
          <Border CornerRadius="10" BorderThickness="1"
           BorderBrush="Black">
            <Border.Background>
              <RadialGradientBrush>
                <RadialGradientBrush.GradientStops>
                  <GradientStop Color="LavenderBlush"
                   Offset="0" />
                  <GradientStop Color="Lavender"
                   Offset="0.2" />
                  <GradientStop Color="LightGray"
                   Offset="0.5" />
                  <GradientStop Color="Gray"
                   Offset="1" />
                </RadialGradientBrush.GradientStops>
              </RadialGradientBrush>
            </Border.Background>
            <ScrollViewer x:Name="PART_ContentHost"
             Focusable="False"
             HorizontalAlignment="Center"/>
          </Border>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
  </Style>
</Window.Resources>
```

▲ **Figure 8-18 :** La modification de la représentation dans le style

En définitive, c'est très simple. Il suffit d'appliquer ce que nous avons vu en début de chapitre mais sur l'attribut `Template`.

Il est également possible d'utiliser des ressources dans un style. La ressource doit bien entendu être définie avant le style.

```
<Window.Resources>
   <RadialGradientBrush x:Key="Fondu">
      <RadialGradientBrush.GradientStops>
         <GradientStop Color="LavenderBlush" Offset="0" />
         <GradientStop Color="Lavender" Offset="0.2" />
         <GradientStop Color="LightGray" Offset="0.5" />
         <GradientStop Color="Gray" Offset="1" />
      </RadialGradientBrush.GradientStops>
   </RadialGradientBrush>

   <Style x:Key="MyTextBox">
      <Setter Property="TextBox.Template">
         <Setter.Value>
            <ControlTemplate>
               <Border CornerRadius="10" BorderThickness="1"
                BorderBrush="Black"
                Background="{StaticResource Fondu}">
                  <ScrollViewer x:Name="PART_ContentHost"
                   Focusable="False"
                   HorizontalAlignment="Center"/>
               </Border>
            </ControlTemplate>
         </Setter.Value>
      </Setter>
   </Style>
</Window.Resources>
```

La même ressource peut alors être également utilisée dans la ListBox.

```
<ListBox Name="lstPays" Canvas.Top="100"
 Canvas.Left="200"
 DisplayMemberPath="NomPays"
 SelectedValue="{Binding Path=Pays}"
 SelectedValuePath="Id">
   <ListBox.Template>
      <ControlTemplate>
         <Border CornerRadius="10" BorderThickness="1"
          BorderBrush="Black"
          Background="{StaticResource Fondu}" >
            <ScrollViewer
             Focusable="False"
             HorizontalAlignment="Center">
               <ItemsPresenter/>
            </ScrollViewer>
         </Border>
      </ControlTemplate>
   </ListBox.Template>
</ListBox>
```

Utiliser les triggers

Les triggers, en français déclencheurs, permettent de définir une action qui sera déclenchée quand certaines conditions seront rencontrées ou lorsqu'un événement survient. Bien que le mécanisme soit différent, on peut comparer les déclencheurs XAML et les événements .NET, qui ont finalement un objectif similaire.

Remarque

Utilisation du terme *trigger* plutôt que *déclencheur*

Le terme déclencheur étant fort peu usité par les développeurs, nous utiliserons dans la suite de ce chapitre le terme anglais *trigger*. Il est fréquent en informatique d'utiliser les termes anglais plutôt que français car ils sont très proches des langages de programmation, et l'analogie entre langage parlé et langage de programmation délimite précisément et directement le sens du mot au contexte spécifique.

Les triggers vont nous permettre de générer un certain dynamisme de nos écrans sans recourir au code .NET. Les triggers peuvent être déclenchés lors de la modification de valeurs des propriétés des objets ou associés aux événements d'un objet.

Les triggers sont généralement intégrés dans des styles. Modifions le style de notre exemple précédent pour souligner le champ actif.

```
<Window.Resources>
   <RadialGradientBrush x:Key="Fondu">
      <RadialGradientBrush.GradientStops>
         <GradientStop Color="LavenderBlush" Offset="0" />
         <GradientStop Color="Lavender" Offset="0.2" />
         <GradientStop Color="LightGray" Offset="0.5" />
         <GradientStop Color="Gray" Offset="1" />
      </RadialGradientBrush.GradientStops>
   </RadialGradientBrush>

   <RadialGradientBrush x:Key="FonduBleu">
      <RadialGradientBrush.GradientStops>
         <GradientStop Color="LightBlue" Offset="0" />
         <GradientStop Color="MediumBlue" Offset="0.5" />
         <GradientStop Color="Blue" Offset="0.7" />
         <GradientStop Color="DarkBlue" Offset="1" />
      </RadialGradientBrush.GradientStops>
   </RadialGradientBrush>

   <Style x:Key="MyTextBox">
      <Setter Property="TextBox.Template">
```

```
        <Setter.Value>
           <ControlTemplate>
              <Border CornerRadius="10" BorderThickness="1"
               BorderBrush="Black" x:Name="Zone"
               Background="{StaticResource Fondu}">
                 <ScrollViewer x:Name="PART_ContentHost"
                  Focusable="False"
                  HorizontalAlignment="Center"/>
              </Border>
              <ControlTemplate.Triggers>
                 <Trigger Property="IsFocused"
                  Value="true">
                    <Setter TargetName="Zone"
                     Property="Background"
                     Value="{StaticResource FonduBleu}" />
                 </Trigger>
              </ControlTemplate.Triggers>
           </ControlTemplate>
        </Setter.Value>
     </Setter>
  </Style>
</Window.Resources>
```

▲ **Figure 8-19 :** Trigger pour souligner le champ actif

Si nous analysons les changements, nous retrouvons d'abord l'ajout d'une nouvelle ressource semblable à notre ressource `Fondu` mais avec des couleurs bleues. Ensuite, dans le style proprement dit, une nouvelle balise (`ControlTemplate.Triggers`) a fait son apparition. C'est dans cette balise que nous allons définir tous les triggers dont nous aurons besoin. Chaque trigger est défini au moyen de la balise `Trigger`. L'attribut `Property` détermine la propriété à laquelle le trigger est associé. L'attribut `Value` détermine quant à lui la valeur que doit avoir la propriété pour que le trigger s'exécute. Il s'agit en fait d'une

condition simple sur la valeur de la propriété. Les actions à réaliser sont placées à l'intérieur de ce nœud.

Remarque

Le terme Template

Comme *trigger*, le terme anglais *template* est très souvent privilégié par rapport à sa traduction française, qui surtout dans ce contexte rend mal le concept sous-jacent. C'est pourquoi nous utiliserons ce terme, qui est de plus utilisé tel quel comme propriété dans XAML.

La traduction habituelle du terme *template* est *modèle*. Toutefois, dans le contexte qui nous occupe, il s'agit de la définition de la forme que doit prendre un contrôle en utilisant le langage de description XAML lui-même. C'est par l'utilisation des template que XAML fait la distinction entre le contenu d'un contrôle (la définition de la classe) et sa représentation à l'écran. Le terme modèle prend malgré tout tout son sens si l'on considère que le contrôle sera présenté selon le modèle que vous définissez.

Attention

Nommage des éléments constituant le *template*

Pour pouvoir dans le trigger influencer les différents composants participant à la réalisation du template, chaque élément, ou du moins ceux sur lesquels une action doit être réalisée, doit être nommé au moyen de la propriété `x:Name`.

Pour appliquer un trigger à un contrôle unique, la technique est fort similaire. Nous pouvons appliquer la même technique pour la `ListBox` de notre exemple.

```
<ListBox Name="lstPays" Canvas.Top="100"
 Canvas.Left="200"
 DisplayMemberPath="NomPays"
 SelectedValue="{Binding Path=Pays}"
 SelectedValuePath="Id">
   <ListBox.Template>
      <ControlTemplate>
         <Border x:Name="Zone"  CornerRadius="10"
          BorderThickness="1"  BorderBrush="Black"
          Background="{StaticResource Fondu}">
            <ScrollViewer
             Focusable="False"
             HorizontalAlignment="Center">
               <ItemsPresenter/>
            </ScrollViewer>
         </Border>
```

```
        <ControlTemplate.Triggers>
           <Trigger
            Property="IsMouseOver"
            Value="true">
              <Setter
               TargetName="Zone"
               Property="Background"
               Value="{StaticResource FonduBleu}" />
           </Trigger>
        </ControlTemplate.Triggers>
     </ControlTemplate>
  </ListBox.Template>
</ListBox>
```

▲ **Figure 8-20 :** Trigger sur une ListBox.

La technique utilisée est identique à celle mise en œuvre dans l'exemple précédent.

Remarque

Trigger sur la présence du curseur de la souris

Ici, au lieu d'associer le trigger à la propriété `IsFocused`, il est associé à la propriété `IsMouseOver`.

Créer une animation

Pour réaliser des animations, XAML met à notre disposition la balise `Storyboard`. Cette balise trouve tout naturellement sa place au sein d'un trigger. La balise `Storyboard` va nous permettre de gérer les actions que nous aurons regroupées en son sein. Les animations sont réalisées en faisant varier les propriétés de l'objet à animer. Ce sont les balises `Animation` qui vont nous permettre de les modifier. Vous devrez utiliser la balise `Animation` associée au type de la propriété que vous voulez modifier, par exemple `ByteAnimation` ou `Int32Animation`.

Prenons un exemple simple, un contrôle de type Border qui s'ouvre lors de l'affichage de la fenêtre. Pour y arriver, nous allons définir un trigger sur l'événement Loaded du contrôle.

*Pour en savoir plus sur la manière de créer un trigger, reportez-vous au chapitre **Utiliser les Triggers** page _Ref133244991109.*

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <StackPanel Margin="20">
      <Border Name="Carre"
       Width="10"
       Height="10"
       Background="Blue">
         <Border.Triggers>
            <EventTrigger RoutedEvent="Border.Loaded">
               <BeginStoryboard>
                  <Storyboard>
                     <DoubleAnimation
                      Storyboard.TargetProperty="Width"
                      From="10" To="200" Duration="0:0:2"/>
                     <DoubleAnimation
                      Storyboard.TargetProperty="Height"
                      From="10" To="200" Duration="0:0:2"/>
                  </Storyboard>
               </BeginStoryboard>
            </EventTrigger>
         </Border.Triggers>
      </Border>
   </StackPanel>
</Page>
```

Le trigger contient une balise BeginStoryboard qui elle-même contient une balise Storyboard. C'est dans cette dernière qu'est définie l'animation. Elle consiste à modifier la largeur et la hauteur du cadre. Celui-ci va varier de la taille 10 à 200 en l'espace de 2 secondes. Dans cet exemple, les animations agissent sur le contrôle dans lequel elles sont définies. Vous aurez probablement besoin d'animer plusieurs contrôles lors d'un même événement. Cela est possible en spécifiant l'attribut Storyboard.TargetName. Pour une animation lors de l'ouverture de la fenêtre, l'idéal est d'associer l'animation à l'événement Page.Loaded et non à Border.Loaded.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <Page.Triggers>
      <EventTrigger RoutedEvent="Page.Loaded">
```

```
        <BeginStoryboard>
           <Storyboard>
              <DoubleAnimation
               Storyboard.TargetName="Carre"
               Storyboard.TargetProperty="Width"
               From="10" To="200" Duration="0:0:2"/>
              <DoubleAnimation
               Storyboard.TargetName="Carre"
               Storyboard.TargetProperty="Height"
               From="10" To="200" Duration="0:0:2"/>
           </Storyboard>
         </BeginStoryboard>
      </EventTrigger>
  </Page.Triggers>
  <StackPanel Margin="20">
    <Border Name="Carre"
     Width="10" Height="10"
     Background="Blue"/>
  </StackPanel>
</Page>
```

Attention

Animations simultanées

Les animations définies dans la balise Storyboard sont exécutées simultanément et non successivement.

Cet exemple est finalement assez simple car la transformation modifie directement la valeur de la propriété de l'objet. Ce n'est malheureusement pas toujours le cas. Pour modifier la couleur du fond, nous ne pouvons pas utiliser simplement la balise ColorAnimation comme ceci :

```
<ColorAnimation
 Storyboard.TargetName="Carre"
 Storyboard.TargetProperty="Background"
 From="Blue" To="Red" Duration="0:0:5"/>
```

Le problème n'est pas évident car l'attribut Background de la balise Border reçoit bien comme valeur Blue. Toutefois, dans l'animation, comme le démontre la balise utilisée, l'attribut modifié est un attribut de type Color alors que l'attribut Background attend un attribut de type Brush. La syntaxe correcte sera donc :

```
<ColorAnimation
 Storyboard.TargetName="Carre"
 Storyboard.TargetProperty="Background.Color"
 From="Blue" To="Red" Duration="0:0:5"/>
```

Dans le même ordre d'idée, il sera parfois nécessaire de préciser complètement la propriété.

```
<ColorAnimation
 Storyboard.TargetName="Carre"
 Storyboard.TargetProperty=
   "(Border.Background).(SolidColorBrush.Color)"
 From="Blue" To="Red" Duration="0:0:5"/>
```

De cette façon, il est clair que la propriété modifiée est la propriété `Color` de l'objet de type `SolidColorBrush` assigné à la propriété `Background` de l'objet de type `Border`.

Si vous préférez, vous avez également la possibilité de décrire vous-même le contenu de la propriété `Background`. Vous pourrez ainsi nommer son contenu et y accéder directement dans l'animation.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <Page.Triggers>
      <EventTrigger RoutedEvent="Page.Loaded">
         <BeginStoryboard>
            <Storyboard>
               <DoubleAnimation
                Storyboard.TargetName="Carre"
                Storyboard.TargetProperty="Width"
                From="10" To="200" Duration="0:0:2"/>
               <DoubleAnimation
                Storyboard.TargetName="Carre"
                Storyboard.TargetProperty="Height"
                From="10" To="200" Duration="0:0:2"/>
               <ColorAnimation
                Storyboard.TargetName="Couleur"
                Storyboard.TargetProperty="Color"
                From="LightGray" To="SlateGray"
                Duration="0:0:5" AutoReverse="true"
                RepeatBehavior="Forever"/>
            </Storyboard>
         </BeginStoryboard>
      </EventTrigger>
  </Page.Triggers>
  <StackPanel Margin="20">
    <Border Name="Carre"
     Width="10"
     Height="10">
      <Border.Background>
        <SolidColorBrush x:Name="Couleur"
```

```
        Color="White"/>
      </Border.Background>
    </Border>
  </StackPanel>
</Page>
```

◄ **Figure 8-21 :** Phases dans l'animation du carré

Attention

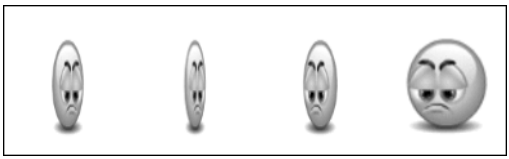
Nommage avec `x:Name`

Pour nommer un sous-élément, vous devez utiliser l'attribut `x:Name` et non l'attribut `Name`.

Jusqu'ici, nous avons utilisé uniquement des animations qui faisaient varier une ou des propriétés d'une valeur à une autre. Il existe une autre méthode permettant de réaliser des animations plus complexes. Cette méthode consiste à passer successivement d'un état à un autre. À titre d'exemple dans notre exercice complet, nous pouvons faire pivoter la photo sur son axe quand la souris passe dessus. Pour y arriver, nous devrons modifier la balise `Border`.

```
<Border Width="100" Height="120"
 BorderThickness="1"
 Background="White" BorderBrush="Black"
 Canvas.Top="10" Canvas.Right="10" >
   <Image Name="Photo"  Source="{Binding Path=Photo}" >
      <Image.RenderTransform>
         <TransformGroup>
            <ScaleTransform ScaleX="1" ScaleY="1"/>
            <TranslateTransform X="0" Y="0"/>
         </TransformGroup>
      </Image.RenderTransform>
      <Image.Triggers>
         <EventTrigger RoutedEvent="Image.MouseMove">
            <EventTrigger.Actions>
               <BeginStoryboard>
                  <Storyboard>
                     <DoubleAnimationUsingKeyFrames
                      Storyboard.TargetProperty=
"(UIElement.RenderTransform).(TransformGroup.Children)[0].(ScaleTransform.ScaleX)">
                        <SplineDoubleKeyFrame Value="-1"
```

```
                        KeyTime="00:00:03"/>
                      <SplineDoubleKeyFrame Value="1"
                        KeyTime="00:00:06"/>
                    </DoubleAnimationUsingKeyFrames>
                    <DoubleAnimationUsingKeyFrames
                     Storyboard.TargetProperty=
                     "(UIElement.RenderTransform)(TransformGroup.Children)
                     ➥ [1].(TranslateTransform.X)">
                      <SplineDoubleKeyFrame Value="100"
                        KeyTime="00:00:03"/>
                      <SplineDoubleKeyFrame Value="0"
                        KeyTime="00:00:06"/>
                    </DoubleAnimationUsingKeyFrames>
                  </Storyboard>
                </BeginStoryboard>
              </EventTrigger.Actions>
            </EventTrigger>
          </Image.Triggers>
        </Image>
</Border>
```



◄ **Figure 8-22 :** Animation tournante

Pour réaliser cette animation, deux transformations sont définies, un changement d'échelle et une translation. Les valeurs définies sont choisies pour que ces transformations n'aient aucune action. Ce n'est que lors de l'activation du trigger que les transformations doivent avoir effectivement lieu.

Renvoi *Pour plus d'informations sur les transformations, reportez-vous au chapitre* ***Appliquer des transformations sur les contrôles*** *page 220.*

Ensuite, liées à l'événement `MouseMove`, deux animations sont définies, une modifiant la transformation d'échelle et l'autre, la translation. Notez que les animations utilisent non pas `DoubleAnimation` mais bien `DoubleAnimationUsingKeyFrames`. Les valeurs à atteindre sont définies dans des nœuds fils : ici des nœuds de type `SplineDoubleKeyFrame`. La valeur à atteindre est spécifiée par l'attribut `Value` et le temps pour y arriver est déterminé par `KeyTime`. Une fois l'état atteint, l'animation passe à l'état suivant. Le passage d'un état à l'autre se fait par interpolation. Cela n'empêche pas les différentes animations d'être réalisées en même temps. Vous devez donc les synchroniser en jouant sur les temps et par exemple en utilisant l'attribut `BeginTime`.

8.4 Checklist

Dans ce chapitre, les notions essentielles que nous avons vues sont :

- comment appliquer une transformation à un contrôle et particulièrement les rotations, les changements d'échelle et les changements d'oblicité ;
- à quoi sert une ressource et comment la créer ;
- comment créer un style ;
- comment utiliser les triggers pour effectuer automatiquement certaines tâches ;
- comment créer simplement des animations.

Les documents

Utiliser FixedDocument 250
Utiliser FlowDocument 254
Éditer un document 275
Annoter un document 282
Checklist 288

Les documents sont au centre des développements informatiques. Ils représentent un moyen efficace de mise à disposition de l'information et sont un complément de plus en plus important des données structurées. WPF, et en ce qui nous concerne plus spécifiquement XAML, met à notre disposition un ensemble d'outils qui vont nous permettre de manipuler et de présenter ce type d'information.

9.1 Utiliser FixedDocument

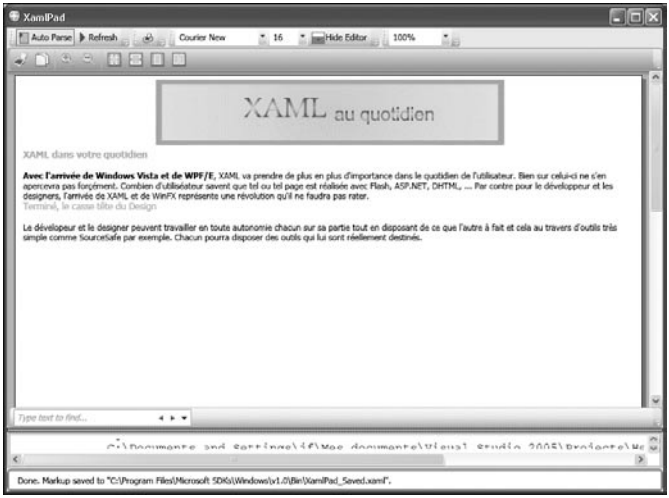
`FixedDocument` permet d'afficher et d'imprimer un document de façon identique quel que soit la définition de l'écran ou de l'imprimante. En outre, ce contrôle offre plusieurs fonctionnalités à l'utilisateur. L'exemple suivant reproduit une page de l'aide de Windows.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <FixedDocument>
      <PageContent Source="Doc1-1.xaml" />
   </FixedDocument>
</Page>
```

Le document *Doc1-1.xaml* :

```
<FixedPage
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <StackPanel HorizontalAlignment="Stretch" >
      <Image>
         <Image.Source>
            Header.gif
         </Image.Source>
      </Image>
      <TextBlock TextWrapping="WrapWithOverflow"
       Margin="5,5,5,5" Width="400"
       HorizontalAlignment="Center">
         <TextBlock Foreground="Orange"
          FontSize="12" FontWeight="Bold">
          XAML en un clin d'œil
         </TextBlock>
         <LineBreak/>
         <LineBreak/>
            Avec Avec l'arrivée de Windows Vista et de
            WPF/E, XAML va prendre de plus en plus
```

```
            d'importance dans le quotidien de
            l'utilisateur. Bien sûr celui-ci ne s'en
            apercevra pas forcément. Combien
            d'utilisateurs savent que telle ou telle page
            est réalisée avec Flash, ASP.NET, DHTML...
            Par contre, pour le développeur et les
            designers, l'arrivée de XAML et de WinFX
            représente une révolution qu'il ne faudra
            pas rater
         <LineBreak/>
         <LineBreak/>
         <TextBlock Foreground="Orange"
          FontSize="12">
            Terminé, le casse-tête du Design
         </TextBlock>
         <LineBreak/>
         <LineBreak/>
            Le développeur et le designer peuvent travailler
            en toute autonomie chacun sur sa partie tout en
            disposant de ce que l'autre a fait et cela au
            travers d'outils très simples comme SourceSafe
            par exemple. Chacun pourra disposer des outils
            qui lui sont réellement destinés.
         <LineBreak/>
         <LineBreak/>
      </TextBlock>
   </StackPanel>
</FixedPage>
```



▲ **Figure 9-1** : Affichage d'un document fixe

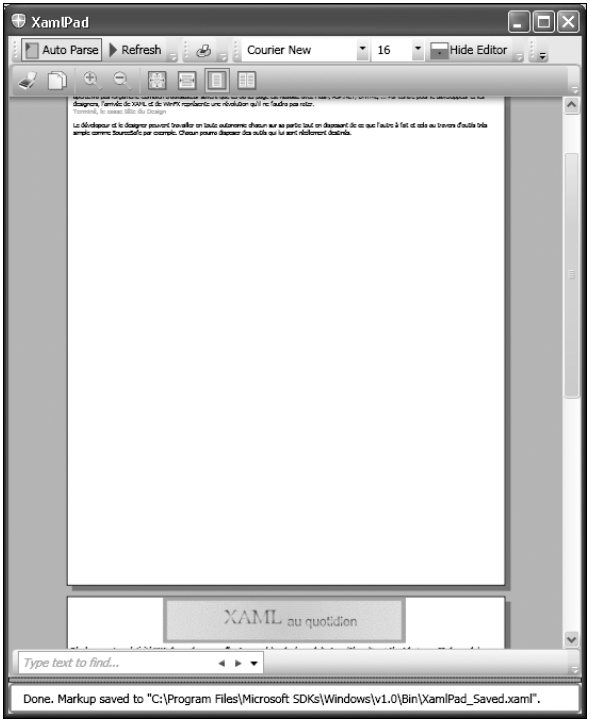
Astuce

Formatage complexe

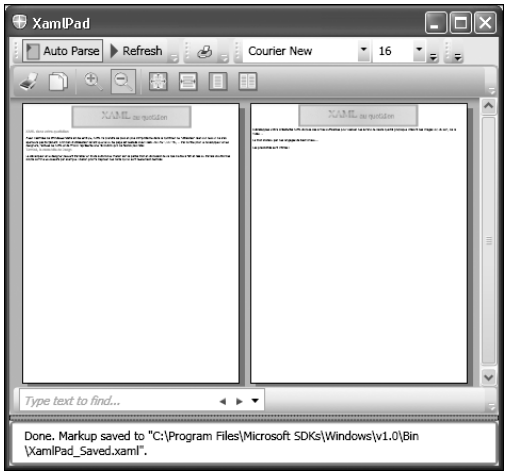
Pour obtenir différents formats complexes de vos caractères, vous pouvez utiliser des TextBlock imbriqués.

Si votre document est composé de plusieurs pages, il suffit d'ajouter de nouvelles balises PageContent pour chaque page du document. Dans l'exemple ci-dessous, vous constaterez qu'il n'est pas nécessaire de créer un fichier par page. Le contenu de la page peut être directement défini dans la balise.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <FixedDocument>
      <PageContent Source="Doc1-1.xaml" />
      <PageContent>
         <FixedPage>
            <StackPanel HorizontalAlignment="Stretch" >
               <Image>
                  <Image.Source>
                     Header.gif
                  </Image.Source>
               </Image>
               <TextBlock TextWrapping="WrapWithOverflow"
                Margin="5,5,5,5" Width="400"
                HorizontalAlignment="Center">
                  <Bold>Développez votre créativité
                  </Bold> Dès…
                  <LineBreak/>
                  <LineBreak/>
                  Encore une autre options sont les
                  programmes…
                  <LineBreak/>
                  <LineBreak/>
                  Les possibilités sont infinies !
               </TextBlock>
            </StackPanel>
         </FixedPage>
      </PageContent>
   </FixedDocument>
</Page>
```



▲ **Figure 9-2 :** Affichage d'un document fixe de plusieurs pages



▲ **Figure 9-3 :** Affichage de deux pages

Si vous souhaitez contrôler certains attributs comme Zoom, ShowPageBorders ou VerticalPageSpacing, vous pouvez inclure votre FixedDocument dans une balise DocumentViewer.

```
<Page xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
   <DocumentViewer Zoom="150">
      <FixedDocument>
…
      </FixedDocument>
   </DocumentViewer>
</Page>
```

9.2 Utiliser FlowDocument

FlowDocument permet d'afficher du texte en adaptant automatiquement sa présentation au mieux selon l'environnement dans lequel il est affiché. L'objectif est donc totalement inverse de FixedDocument.

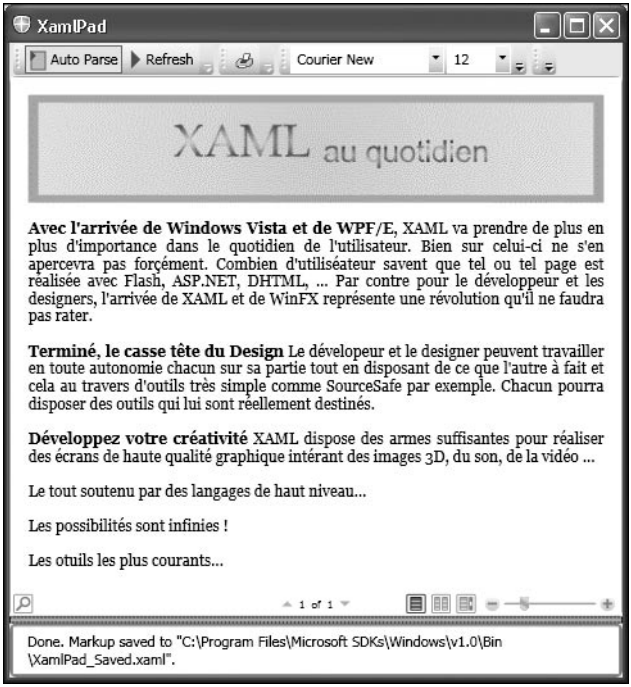
Remarque

Balise obsolète

Vous rencontrerez peut-être au détour d'Internet des exemples de documents utilisant la balise TextFlow. Toutefois, sachez que cette possibilité a été retirée à partir de la version bêta 2 de WinFX et ne peut donc plus être utilisée.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <FlowDocument TextAlignment="Justify">
      <Paragraph>
         <Image>
            <Image.Source>
               Photo.jpg
            </Image.Source>
         </Image>
      </Paragraph>
      <Paragraph>
         <Bold>Développez votre créativité</Bold>
         Dès que vous…
```

```
      </Paragraph>
      <Paragraph>
         Encore une autre…
      </Paragraph>
      <Paragraph>
         Les possibilités sont infinies !
      </Paragraph>
   </FlowDocument>
</Page>
```



▲ **Figure 9-4 :** Affichage d'une page avec FlowDocument

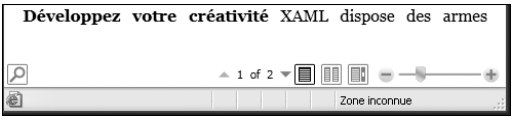
Attention

Association à la classe

Pour rappel, si vous utilisez XAMLPad, l'attribut `x:Class="Page1"` doit être retiré du code puisque aucun code *behind* (.Net) n'est associé.

◄ **Figure 9-5 :** Le même document

Notez que, lorsque vous redimensionnez la fenêtre, `FlowDocument` ne génère pas de défilement mais fait une gestion de pages. Dans l'exemple ci-dessus, le document occupe maintenant deux pages. La position courante et le nombre de pages sont indiqués dans la barre d'outils en bas du document.



▲ **Figure 9-6 :** Le nombre de pages

Vous pouvez naviguer entre les pages en utilisant les petites flèches dans la barre d'outils. Vous pouvez également, depuis cette barre d'outils, choisir l'affichage sur deux pages côte à côte ou opter pour un défilement continu.

Cette barre d'outils contient également un `Slider` qui vous permet de zoomer sur le document. Le zoom influence automatiquement le nombre de pages.

Vous pouvez encore imposer des sauts de page en utilisant les attributs `BreakPageBefore` ou `BreakPageAfter`.

Ajoutez le code suivant à la fin du document.

```
<Paragraph BreakPageBefore="True">
   Nouvelle page
</Paragraph>
```

Quelle que soit la taille de la fenêtre, *Nouvelle page* sera toujours sur une page séparée. Cet attribut s'applique aussi bien à `Paragraph` que `List` et même `Section`. Il est d'ailleurs temps de vous en dire un peu plus sur la balise `Section`. Celle-ci a pour but de regrouper un certain nombre d'éléments du texte exactement comme une section en Word. Il devient alors possible d'appliquer certains attributs sur l'ensemble des paragraphes contenus dans la section.

```
<Section BreakPageBefore="True" FontSize="24">
   <Paragraph>
      Nouvelle page
   </Paragraph>
   <Paragraph>
      Ce texte est dans la même
      section que "Nouvelle page"
   </Paragraph>
</Section>
```

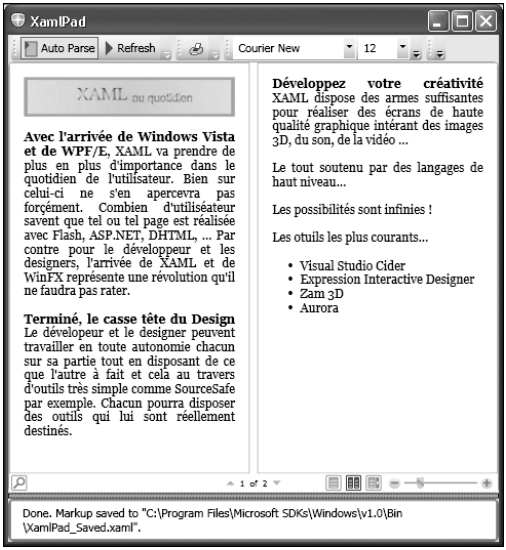
Outre la balise `Section` et la balise `Table`, dont nous parlerons plus loin dans le chapitre, vous pouvez utiliser au sein de `FlowDocument` la balise `List`.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <FlowDocument TextAlignment="Justify">
…
         <Paragraph>
            Les moyens les plus courants…
         </Paragraph>
         <List>
            <ListItem>
               <Paragraph>
                  Appreil photo
               </Paragraph>
            </ListItem>
```

Chaque élément de la liste est défini dans un objet `ListItem` séparé.

```
            <ListItem>
               <Paragraph>
                  Téléphone portable
               </Paragraph>
            </ListItem>
            <ListItem>
               <Paragraph>
```

```
                Camescope
            </Paragraph>
        </ListItem>
        <ListItem>
            <Paragraph>
                Webcam
            </Paragraph>
        </ListItem>
    </List>
  </FlowDocument>
</Page>
```



▲ **Figure 9-7 :** Une liste dans un FlowDocument

Remarque

Attributs de `FlowDocument`

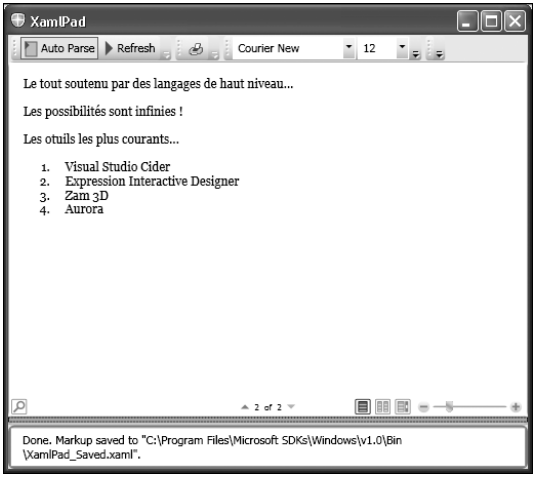
Afin d'améliorer la présentation, vous pouvez également utiliser des attributs déjà vus précédemment pour d'autres contrôles tels que `TextAlignment`.

La balise `List` peut être configurée pour réaliser les listes à puce les plus courantes ou même des listes numérotées.

Avec l'attribut `MarkerStyle`, vous allez pouvoir choisir le type de puce ou de numéro. Les différentes puces possibles sont `Disk`, `Circle`, `Square` et `Box` alors

que, pour les listes numérotées, vous pouvez utiliser `LowerLatin`, `UpperLatin`, `LowerRoman`, `UpperRoman` ou encore `Decimal`. L'attribut `MarkerOffset` détermine l'espace entre le texte et la puce. Si vous optez pour une liste numérotée, vous pouvez influencer le numéro d'origine en utilisant l'attribut `StartIndex`.

```
<List StartIndex="1" MarkerStyle="Decimal" MarkerOffset="20">
```



▲ **Figure 9-8 :** Une liste numérotée dans un FlowDocument

Si les possibilités offertes par `List` ne vous satisfont pas, c'est alors le moment de voir le `BulletDecorator`. Ce n'est pas un contrôle spécifique au document mais c'est certainement une des bonnes façons de l'utiliser.

```
<Page xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:sys="clr-namespace:System;assembly=mscorlib"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <StackPanel>
      <TextBlock>
         BulletDecorator permet d'afficher des listes
         personnalisées.
      </TextBlock>
      <BulletDecorator>
         <BulletDecorator.Bullet>
            <Image Height="16" Source="c:\bullet.gif"/>
         </BulletDecorator.Bullet>
         <Label>
            Photographie
         </Label>
      </BulletDecorator>
```

Le BulletDecorator est divisé en deux parties. La première, définie dans la propriété Bullet, détermine la forme de la puce. La seconde partie définit la forme que doit prendre le texte. Il va sans dire que rien ne vous oblige à vous limiter à une image et à du texte.

```
      <BulletDecorator Margin="20,0,0,0">
         <BulletDecorator.Bullet>
            <Image Height="16" Source="c:\bullet.jpg"/>
         </BulletDecorator.Bullet>
         <Label>
            Film
         </Label>
      </BulletDecorator>
```

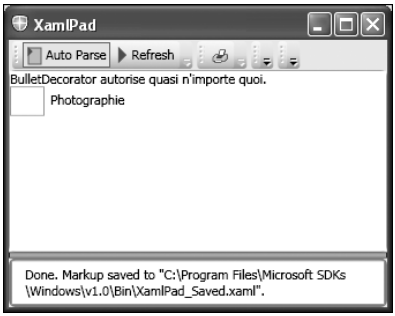
En modifiant la marge, vous pouvez ajuster l'indentation de votre élément. Notez que chaque élément peut prendre une forme différente. Contrairement aux ListItem, il s'agit en fait chaque fois d'un contrôle séparé indépendant des autres.

```
      <BulletDecorator Margin="20,0,0,0">
         <BulletDecorator.Bullet>
            <Image Height="16" Source="c:\bullet.jpg"/>
         </BulletDecorator.Bullet>
         <Label>
            Digitale
         </Label>
      </BulletDecorator>
      <BulletDecorator>
         <BulletDecorator.Bullet>
            <Image Height="16" Source="c:\bullet.gif"/>
         </BulletDecorator.Bullet>
         <Label>
            Vidéo
         </Label>
      </BulletDecorator>
   </StackPanel>
</Page>
```

◄ **Figure 9-9** : Une liste à puce réalisée avec BulletDecorator

Avec l'exemple suivant, nous pouvons voir que bien d'autres contrôles peuvent être inclus dans le `BulletDecorator`.

```
<Page xmlns=
 http://schemas.microsoft.com/winfx/2006/xaml/presentation
 xmlns:sys="clr-namespace:System;assembly=mscorlib"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <StackPanel>
      <TextBlock>
         BulletDecorator autorise quasi n'importe quoi.
      </TextBlock>
      <BulletDecorator>
         <BulletDecorator.Bullet>
            <TextBox Width="30"/>
         </BulletDecorator.Bullet>
         <Label>
            Photographie
         </Label>
      </BulletDecorator>
   </StackPanel>
</Page>
```



◄ **Figure 9-10** : Un BulletDecorator pour le moins original

Comme vous pouvez le constater, `BulletDecorator` permet le meilleur mais aussi le pire. À vous de l'utiliser à bon escient.

Après ce petit détour, revenons au sujet qui nous occupe, l'affichage d'un document. Parfois, le document ne sera qu'un élément de votre page parmi d'autres. Vous souhaiterez alors peut-être le placer dans une grille. Malheureusement, `FlowDocument` ne peut être inclus tel quel dans un contrôle de type `Grid` ou `StackPanel`. En revanche, il est possible de l'inclure dans une balise `FlowDocumentScrollViewer`.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <FlowDocumentScrollViewer>
      <FlowDocumentScrollViewer.Document>
         <FlowDocument>
…
         </FlowDocument>
      </FlowDocumentScrollViewer.Document>
   </FlowDocumentScrollViewer>
</Page>
```

Comme vous pouvez le constater, la présentation du document est différente une fois qu'il est inclus dans ce contrôle.

◄ **Figure 9-11 :** Un document dans un FlowDocumentScrollViewer

La barre d'outils ne présente plus la navigation entre les pages. En fait, la notion de page disparaît. C'est pourquoi une barre de défilement fait son apparition sur la droite.

Remarque

`BreakPageBefore`

Malgré la présence de l'attribut `BreakPageBefore`, le texte se suit sans aucun saut de page.

Il reste en revanche la possibilité de faire un zoom.

Le FlowDocumentScrollViewer peut être intégré dans un conteneur de type grille, par exemple.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <FlowDocumentScrollViewer Grid.Column="1">
         <FlowDocumentScrollViewer.Document>
            <FlowDocument>
…
            </FlowDocument>
         </FlowDocumentScrollViewer.Document>
      </FlowDocumentScrollViewer>
   </Grid>
</Page>
```

▲ **Figure 9-12** : Le même inclus dans une grille

Il en va de même pour FlowDocumentPageViewer.

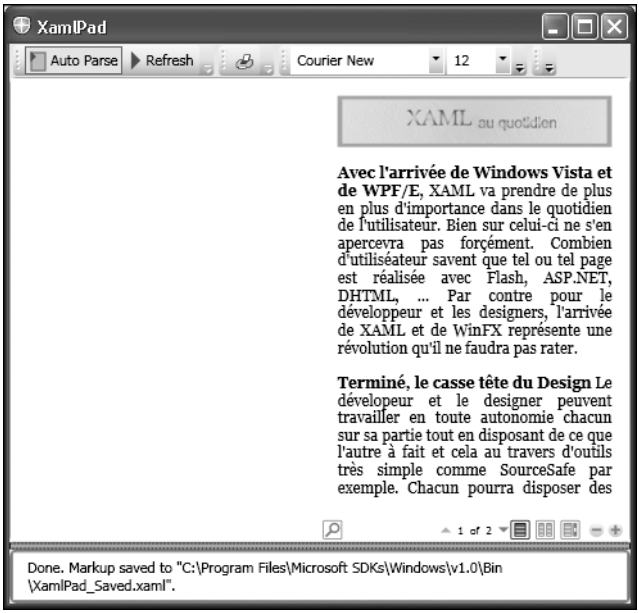
```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <FlowDocumentPageViewer Grid.Column="1">
         <FlowDocumentPageViewer.Document>
            <FlowDocument>
…
            </FlowDocument>
         </FlowDocumentPageViewer.Document>
      </FlowDocumentPageViewer>
   </Grid>
</Page>
```

▲ **Figure 9-13 :** Avec un FlowDocumentPageViewer

Il est possible de regrouper les avantages de ces deux techniques d'affichage d'un document en mode Flow en utilisant la balise FlowDocumentReader. C'est par ailleurs elle qui est utilisée par défaut si vous n'incluez pas FlowDocument dans une des deux autres balises. Toutefois, il doit être explicitement défini si

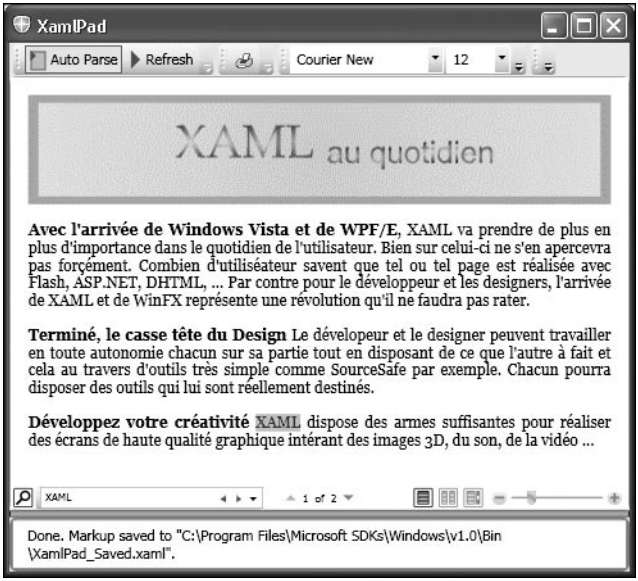
vous désirez par exemple intégrer votre FlowDocument dans une grille.

```
<Page x:Class="Page1"
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1"
 >
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <FlowDocumentReader>
         <FlowDocumentReader.Document>
            <FlowDocument>
…
            </FlowDocument>
         </FlowDocumentReader.Document>
      </FlowDocumentReader>
   </Grid>
</Page>
```



▲ **Figure 9-14 :** Utiliser FlowDocumentReader

La gestion des pages fait à nouveau son apparition dans la barre d'outils sous le document. Comme nous l'avions vu sans le savoir, avec ce contrôle l'utilisateur peut choisir, grâce aux boutons dans cette barre d'outils, entre l'affichage par une ou deux pages mais aussi en mode de défilement. Une loupe sur la gauche permet d'ouvrir une extension de la barre d'outils afin de réaliser des recherches dans le texte.



▲ **Figure 9-15 :** Recherche dans le texte

Les possibilités de mise en page ne s'arrêtent pas là. Vous pouvez par exemple insérer une figure dans votre document. Une figure permet d'introduire du contenu à un endroit spécifique de la page.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Figure"
 >
   <FlowDocument>
      <Paragraph>
         <Figure HorizontalAnchor="ContentLeft">
```

La figure peut contenir autre chose qu'une image. Notez que, même avec une image, vous devez l'inclure dans un paragraphe.

```
            <Paragraph FontSize="8">
               <Image Width="40" Source="idea.gif"/>
               <LineBreak/>
               Idée.
            </Paragraph>
         </Figure>
         <Bold>Une idée !</Bold>
         Pourquoi ne pas placer une figure dans votre
         texte. Cela peut être du plus bel effet dans
         la présentation d'un document.
      </Paragraph>
      <Paragraph>
         Vous venez encore de découvrir une nouvelle
         possibilité avec XAML.
      </Paragraph>
   </FlowDocument>
</Page>
```

◄ **Figure 9-16** : Une figure dans le texte

Comme vous pouvez le constater, la figure se place en parallèle du texte. Si nous avions simplement ajouté notre image dans le texte, le résultat aurait été fort différent.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Image"
```

```
 >
   <FlowDocument>
      <Paragraph>
         <Image Width="40" Source="idea.gif"/>
         <Bold>Une idée !</Bold>
         Pourquoi ne pas placer une figure dans votre
         texte. Cela peut être du plus bel effet dans
         la présentation d'un document.
      </Paragraph>
      <Paragraph>
         Vous venez encore de découvrir une nouvelle
         possibilité avec XAML.
      </Paragraph>
   </FlowDocument>
</Page>
```

◄ **Figure 9-17** : Une image dans le texte

Comme toujours, vous disposez d'un certain nombre d'attributs pour modifier le comportement de la figure. Vous avez déjà pu voir `HorizontalAnchor` dans l'exemple précédent. Il existe également `VerticalAnchor`. Remplacez dans l'exemple la balise `Figure` par celle ci-dessous.

```
<Figure HorizontalAnchor="ContentRight"
 VerticalAnchor="ContentBottom">
```

◄ **Figure 9-18 :** Modification des ancres de la figure

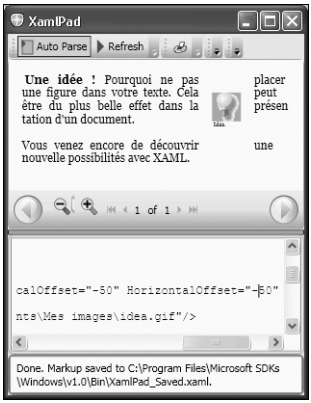
Attention

Position de la figure

Comme vous pouvez le constater, la figure se positionne non pas en fonction du paragraphe dans lequel elle est incluse mais bien en fonction de la page.

Les attributs `VerticalOffset` et `HorizontalOffset` vont également vous permettre de modifier la position en produisant un décalage. Le décalage peut être une valeur positive ou négative selon le sens désiré.

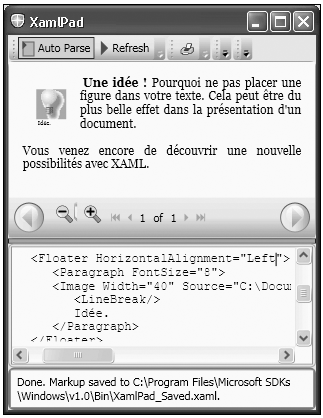
```
<Figure HorizontalAnchor="ContentRight"
 VerticalAnchor="ContentBottom"
 VerticalOffset="-50" HorizontalOffset="-50">
```



◄ **Figure 9-19 :** Modification des offsets de la figure

Outre `Figure`, vous disposez également de la balise `Floater` pour réaliser la présentation du document. `Floater` est très semblable à `Figure` mais, contrairement à ce dernier, il se positionne par rapport au flux dans lequel il est inclus. L'attribut principal de `Floater` est `HorizontalAlignment`, qui va permettre de positionner le contenu à gauche, à droite ou même au centre du reste du flux.

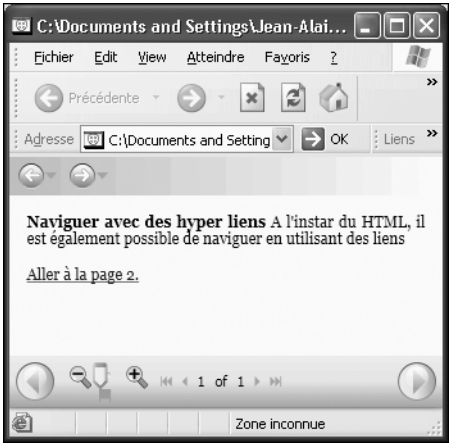
```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Floater"
 >
   <FlowDocument>
      <Paragraph>
         <Floater HorizontalAlignment="Left">
            <Paragraph FontSize="8">
               <Image Width="40" Source="idea.gif"/>
               <LineBreak/>
               Idée.
            </Paragraph>
         </Floater>
         <Bold>Une idée !</Bold>
         Pourquoi ne pas placer une figure dans votre
         texte. Cela peut être du plus bel effet dans
         la présentation d'un document.
      </Paragraph>
      <Paragraph>
         Vous venez encore de découvrir une nouvelle
         possibilité avec XAML.
      </Paragraph>
   </FlowDocument>
</Page>
```



◀ **Figure 9-20 :** Utilisation d'un Floater

Une dernière fonctionnalité intéressante est la navigation par les liens. Cette navigation fonctionne comme les hyperliens du HTML. Elle vous permet d'atteindre une autre page XAML simplement en cliquant sur le texte.

```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Hyperlien"
 >
   <FlowDocument>
      <Paragraph>
         <Bold>Naviguer avec des hyperliens</Bold>
         A l'instar du HTML, il est également possible
         de naviguer en utilisant des liens
      </Paragraph>
      <Paragraph>
         <Hyperlink NavigateUri="Page2.xaml">
            Aller à la page 2.
         </Hyperlink>
      </Paragraph>
   </FlowDocument>
</Page>
```

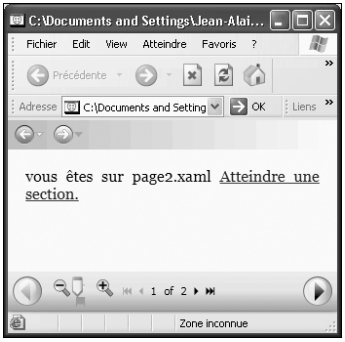


◄ **Figure 9-21** : Le premier document

Si vous cliquez sur le lien, le second document est chargé.

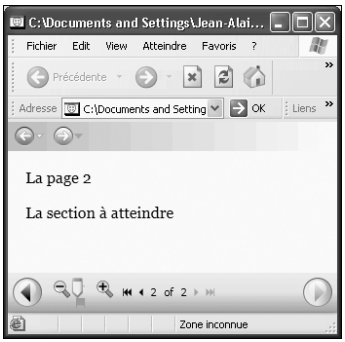
```
<Page
 xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Hyperlien et section"
```

```
>
  <FlowDocument>
    <Paragraph>
      vous êtes sur  page2.xaml
      <Hyperlink NavigateUri="#suite">
        Atteindre une section.
      </Hyperlink>
    </Paragraph>
    <Paragraph BreakPageBefore="True">
      La page 2
    </Paragraph>
    <Section Name="suite">
      <Paragraph>
        La section à atteindre
      </Paragraph>
    </Section>
  </FlowDocument>
</Page>
```



◄ **Figure 9-22 :** Le second document

Notez que ce second document possède deux pages. En cliquant sur le lien, vous atteignez la section présente sur la deuxième page.



◄ **Figure 9-23 :** La section

Attention

Notion de page

Quand on parle de page, il faut toujours bien faire la distinction entre page du navigateur et page du document. Il s'agit de notions bien distinctes.

Bien que les hyperliens soient parfaitement adaptés à l'utilisation dans des documents, il est tout à fait possible de les placer dans des contrôles plus traditionnels comme un `Label`.

Pour pouvoir disposer de toutes les fonctionnalités nécessaires à l'affichage d'un document, il nous manque encore un grand classique, le tableau. Pour générer un tableau, nous disposons de toute une batterie de balises, à commencer par la balise `Table`. Un peu à la façon d'une grille, nous devons commencer par définir le nombre de colonnes. Les colonnes sont définies au sein d'un nœud `Table.Columns` et décrites au moyen de la balise `TableColumn`. En revanche, les lignes sont définies au fur et à mesure avec la balise `TableRow` et sont incluses dans un nœud `TableRowGroup`. Chaque ligne contient un ensemble de cellules. Les cellules sont créées par la balise `TableCell` et prennent place successivement dans les colonnes. Il n'est pas nécessaire de définir autant de cellules que de colonnes ; toutefois, si une colonne doit rester vide mais qu'au moins une colonne suivante est remplie, vous devrez créer une cellule vide pour cette colonne. À l'intérieur des cellules, vous pouvez placer n'importe quel contenu et même éventuellement un autre tableau. À titre d'exemple, créons un tableau simple.

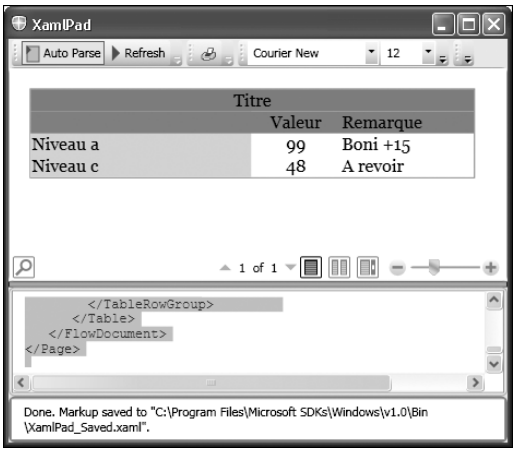
```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 >
   <FlowDocument>
      <Table BorderBrush="Gray" BorderThickness="1">
         <Table.Columns>
            <TableColumn Width="200"
             Background="LightGray"/>
            <TableColumn Width="80"/>
            <TableColumn Width="120"/>
         </Table.Columns>
```

Après avoir défini les colonnes, vous devez définir les lignes et leur contenu.

```
         <TableRowGroup>
            <TableRow Background="SlateGray">
               <TableCell
                ColumnSpan="3"
```

```
            Block.TextAlignment="Center">
              <Paragraph>
                 Titre
              </Paragraph>
           </TableCell>
        </TableRow>
        <TableRow Background="Gray">
           <TableCell/>
           <TableCell
            Block.TextAlignment="Center">
              <Paragraph>
                 Valeur
              </Paragraph>
           </TableCell>
           <TableCell
            Block.TextAlignment="Left">
              <Paragraph>
                 Remarque
              </Paragraph>
           </TableCell>
        </TableRow>
        <TableRow>
           <TableCell>
              <Paragraph>
                 Niveau a
              </Paragraph>
           </TableCell>
           <TableCell
            Block.TextAlignment="Center">
              <Paragraph>
                 99
              </Paragraph>
           </TableCell>
           <TableCell>
              <Paragraph>
                 Boni +15
              </Paragraph>
           </TableCell>
        </TableRow>
        <TableRow>
           <TableCell>
              <Paragraph>
                 Niveau c
              </Paragraph>
           </TableCell>
           <TableCell
            Block.TextAlignment="Center">
              <Paragraph>
                 48
              </Paragraph>
```

```
            </TableCell>
            <TableCell>
               <Paragraph>
                  A revoir
               </Paragraph>
            </TableCell>
         </TableRow>
      </TableRowGroup>
   </Table>
</FlowDocument>
</Page>
```



▲ **Figure 9-24** : Affichage d'un tableau

Notez dans cet exemple la présence de l'attribut `Block.TextAlignment`, qui va permettre de spécifier l'alignement du contenu de la cellule.

9.3 Éditer un document

Le meilleur moyen pour éditer un document reste le `RichTextBox`. Bien sûr, comme son prédécesseur, il reste un contrôle brut qui dispose en interne de tous les mécanismes nécessaires mais pas d'outil standard pour les exposer à l'utilisateur final. Si vous désirez le transformer en un vrai éditeur, il vous reste beaucoup de travail à fournir.

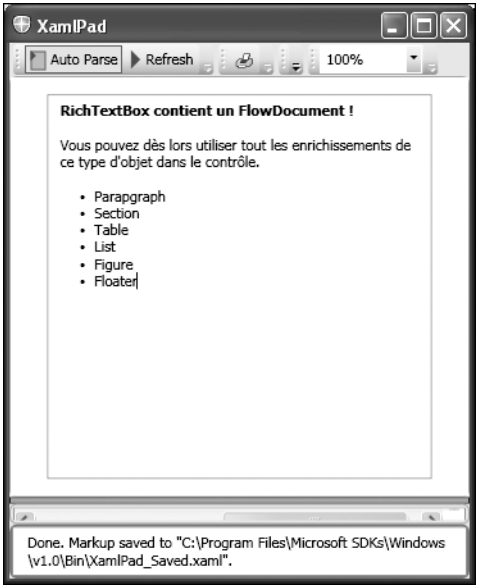
```
<Page
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="testrtf" Height="300" Width="300"
 >
```

```
   <Grid>
      <RichTextBox>
```

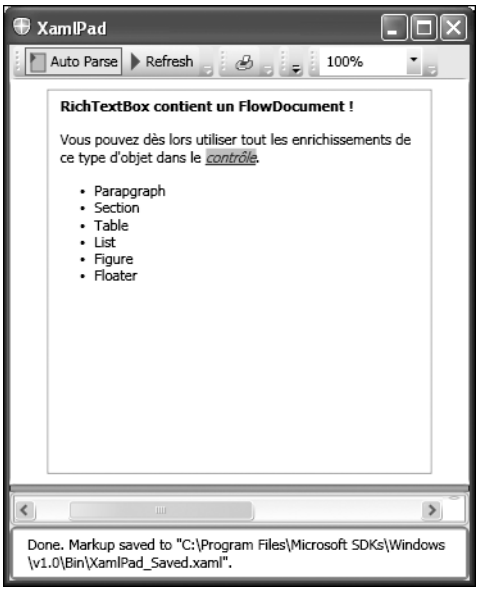
En XAML, vous pouvez intégrer un `FlowDocument` dans le control `RichTextBox`.

```
         <FlowDocument>
            <Paragraph>
               <Bold>
                  RichTextBox contient un FlowDocument !
               </Bold>
            </Paragraph>
            <Paragraph>
               Vous pouvez dès lors utiliser tous les
               enrichissements de ce type d'objet dans
               le contrôle.
            </Paragraph>
            <List>
               <ListItem>
                  <Paragraph>Parapgraph</Paragraph>
               </ListItem>
               <ListItem>
                  <Paragraph>Section</Paragraph>
               </ListItem>
               <ListItem>
                  <Paragraph>Table</Paragraph>
               </ListItem>
               <ListItem>
                  <Paragraph>List</Paragraph>
               </ListItem>
               <ListItem>
                  <Paragraph>Figure</Paragraph>
               </ListItem>
               <ListItem>
                  <Paragraph>Floater</Paragraph>
               </ListItem>
            </List>
         </FlowDocument>
      </RichTextBox>
   </Grid>
</Page>
```

En utilisant les touches d'édition, vous pouvez enrichir le format du texte dactylographié. Par exemple, Ctrl+U pour souligner ou Ctrl+I pour mettre le texte en italique. Vous trouverez la liste complète des raccourcis dans les annexes.



◄ **Figure 9-25 :** Éditer un FlowDocument



◄ **Figure 9-26 :** Enrichir le format

Dans la pratique, il sera très rare de modifier un document fixé dans le code. Mais, plus vraisemblablement, vous devrez charger et sauver le document

dynamiquement. À titre d'exemple, nous pouvons réaliser un menu contextuel qui permettra le chargement et l'enregistrement d'un document mais également son impression. Deux options complémentaires permettront de voir d'une part comment ajouter du contenu riche et d'autre part comment utiliser les commandes d'édition autrement que *via* les touches de raccourci.

Nous ajoutons un simple menu contextuel dans le code XAML avec les appels aux méthodes qui y sont associées.

```
<Window x:Class="Window1"
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="testrtf" Height="300" Width="300"
 >
   <Grid>
      <RichTextBox Name="rtfDocument">
         <RichTextBox.ContextMenu>
            <ContextMenu>
               <MenuItem Click="Charger"
                Header="Charger"/>
               <MenuItem Click="Sauver"
                Header="Sauver"/>
               <Separator/>
               <MenuItem Click="AddButton"
                Header="ajout"/>
               <Separator/>
               <MenuItem Click="Imprimer"
                Header="Imprimer"/>
               <Separator/>
```

Pour faire appel à une commande d'édition, nul besoin de faire appel à du code .NET, il suffit d'assigner la commande voulue à la propriété `Command`.

```
               <MenuItem
                Command="EditingCommands.ToggleBullets"
                Header="Puces"/>
            </ContextMenu>
         </RichTextBox.ContextMenu>
      </RichTextBox>
   </Grid>
</Window>
```

Ensuite, nous devons adapter le code .NET en conséquence.

```
' Interaction logic for Window1.xaml
Imports System.IO
Partial Public Class Window1
   Inherits Window
```

```
Public Sub New()
   InitializeComponent()
End Sub
```

En premier, voyons comment sauver le contenu de notre RichTextControl. La première chose à faire est de définir un TextRange englobant le contenu complet du document. Ensuite, grâce à ce TextRange, nous pourrons sauver le document dans un fichier *via* un FileStream, et ce dans le format désiré. Vous avez le choix entre le format RTF, bien sûr, mais aussi XAML et d'autres également.

```
Public Sub Sauver(ByVal sender As Object _
                , ByVal e As RoutedEventArgs)
   Dim range As TextRange
   Dim fichier As FileStream
   range = New TextRange( _
                  rtfDocument.Document.ContentStart _
                  , rtfDocument.Document.ContentEnd)
   fichier = New FileStream("Sauvegarde.xml" _
                         , FileMode.Create)
   range.Save(fichier, DataFormats.Xaml)
   fichier.Close()
End Sub
```

Pour récupérer le document, il suffit d'appliquer la même méthode mais en sens inverse avec la méthode Load de la classe TextRange.

```
Public Sub Charger(ByVal sender As Object _
                , ByVal e As RoutedEventArgs)
   Dim range As TextRange
   Dim fichier As FileStream
   If File.Exists("Sauvegarde.xml") Then
      range = New TextRange( _
                  rtfDocument.Document.ContentStart _
                  , rtfDocument.Document.ContentEnd)
      fichier = New FileStream("Sauvegarde.xml" _
                            , FileMode.Open)
      range.Load(fichier, DataFormats.Xaml)
      fichier.Close()
   End If

End Sub
```

Pour imprimer un document, le plus simple est d'utiliser la classe PrintDialog, qui est capable d'imprimer les objets dont la classe hérite de Visual, ce qui est justement le cas de FlowDocument, comme c'est par ailleurs le cas de tous les contrôles UIElement et de tous les conteneurs.

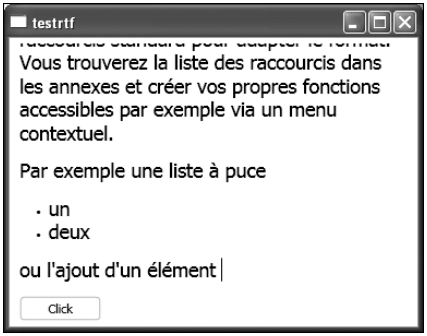
```
    Public Sub Imprimer(ByVal sender As Object _
                        , ByVal e As RoutedEventArgs)
        Dim choixImprimante As New PrintDialog()
        If choixImprimante.ShowDialog() Then
            choixImprimante.PrintVisual( _
                                DirectCast(rtfDocument, Visual) _
                                , "Impression FlowDocument")
        End If

    End Sub
```

Pour ajouter du contenu complexe comme un bouton, il est d'abord nécessaire de créer le contenu à insérer, ce qui est fait par `New Button` ; ensuite, ce contenu est ajouté par exemple dans un paragraphe et placé dans le document par l'intermédiaire de la propriété `Blocks`.

```
    Public Sub AddButton(ByVal sender As Object _
                         , ByVal e As RoutedEventArgs)

        Dim elem As New Button
        elem.Content = "Click"

        rtfDocument.Document.Blocks.Add(New _
                    Paragraph((New InlineUIContainer(elem))))

    End Sub

End Class
```

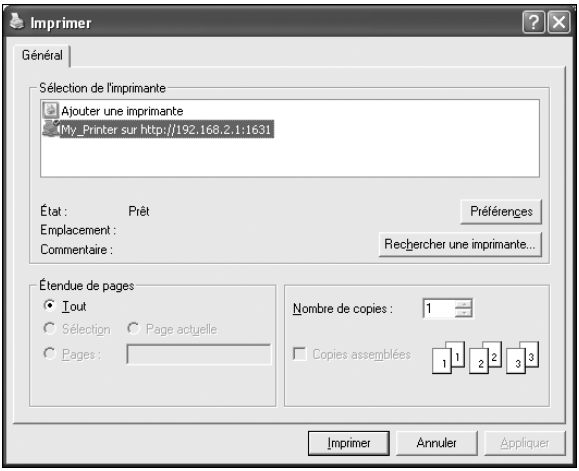
◄ **Figure 9-27** : Le RichTextBox et son menu contextuel

Le même contrôle après usage de l'ajout de contenu riche et de l'utilisation des puces.



◄ **Figure 9-28** : Le RichTextBox et son menu contextuel

Lorsque vous demandez l'impression, vous recevez automatiquement l'écran standard de choix d'imprimante.

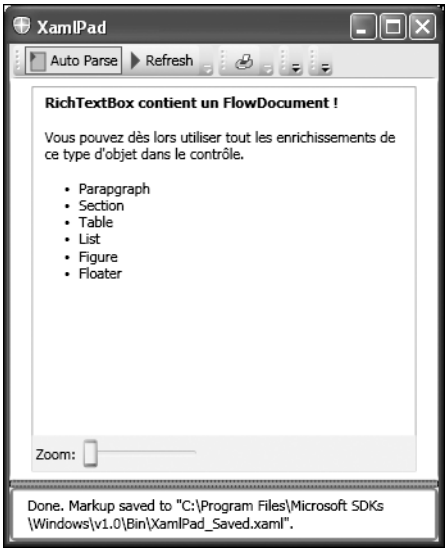


▲ **Figure 9-29** : L'écran standard d'impression

Si vous souhaitez aller plus loin dans cette voie et créer sur cette base un éditeur complexe, vous aurez certainement l'usage d'une barre de statut. Cette possibilité n'est bien sûr pas limitée à l'usage d'un `RichTextBlock` et peut être utilisée dans de nombreux contextes. Son usage se révèle relativement simple. Il suffit de la définir à l'intérieur du conteneur dans lequel elle doit prendre place. Les éléments qui doivent être présents sont placés dans des `StatusBarItem`.

```
<Page xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="testrtf" Height="300" Width="300"
 >
```

```
   <Grid>
      <RichTextBox>
…
      </RichTextBox>
      <StatusBar Grid.Column="0" Grid.Row="1"
       VerticalAlignment="Bottom" Background="Beige">
         <StatusBarItem>
            <TextBlock>
               Zoom:
            </TextBlock>
         </StatusBarItem>
         <StatusBarItem>
            <Slider/>
         </StatusBarItem>
      </StatusBar>
   </Grid>
</Page>
```



◄ **Figure 9-30** : Une barre de statut

Vous pouvez placer dans la barre de statut tous les contrôles dont vous aurez besoin.

9.4 Annoter un document

WPF offre un système complet permettant d'annoter aussi bien les documents fixes que les documents en mode flux. Les annotations pourront être sauvées dans une table SQL ou plus simplement dans un fichier XML.

La première chose à voir dans le code ci-dessous, c'est la présence d'un namespace supplémentaire. Celui-ci reçoit comme nom ann. Notez aussi la présence de l'événement Closed.

```
<Window x:Class="Window1"
 xmlns=
⎵"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:ann=
⎵"clr-namespace:System.Windows.Annotations;assembly=PresentationFramework"
 Title="Les annotations" Height="500" Width="500"
 Closed="OnClose"
 >
   <StackPanel>
```

Un menu où seront reprises les fonctionnalités des annotations est ajouté à la fenêtre. Ainsi que vous pouvez le constater, nous utilisons comme pour l'édition d'un document des commandes prédéfinies, mais cette fois en provenance d'AnnotationService. Ces fonctionnalités sont au nombre de six :

- surligner la sélection ;
- associer une note à la sélection ;
- associer une note dessinée à la sélection ;
- enlever le surlignement de la sélection ;
- effacer les notes de la sélection ;
- effacer les notes et les surlignements dans la sélection.

```
      <Menu Name="MainMenu" >
         <MenuItem Header="Annotations" >
         <MenuItem Command=
          "ann:AnnotationService.CreateHighlightCommand"
          Header="Surligner" />
         <MenuItem Command=
          "ann:AnnotationService.CreateTextStickyNoteCommand"
          Header="Nouvelle note" />
         <MenuItem Command=
          "ann:AnnotationService.CreateInkStickyNoteCommand"
          Header="Nouvelle note manuscrite" />
         <Separator />
         <MenuItem Command=
          "ann:AnnotationService.ClearHighlightsCommand"
          Header="Enlever le surlignement" />
         <MenuItem Command=
          "ann:AnnotationService.DeleteStickyNotesCommand"
          Header="Effacer les notes" />
         <MenuItem Command=
```

```
          "ann:AnnotationService.DeleteAnnotationsCommand"
          Header="Effacer notes et surlignement" />
      </Menu>
```

Ensuite, il ne reste qu'à définir notre document.

```
      <FlowDocumentReader Name="docViewer">
         <FlowDocument TextAlignment="Justify">
            <Paragraph>
               <Image>
                  <Image.Source>
                     c:\Photo.jpg
                  </Image.Source>
               </Image>
            </Paragraph>
            <Paragraph>
               <Bold>XAML en un clin d'œil</Bold>
            </Paragraph>
            <Paragraph>
               Avec Avec l'arrivée de Windows Vista et de…
            </Paragraph>
            <Paragraph>
              <Bold>
                 Terminé, le casse-tête du Design
              </Bold>
            </Paragraph>
            <Paragraph>
               Le développeur et le designer peuvent…
            </Paragraph>
         </FlowDocument>
      </FlowDocumentReader>
   </StackPanel>
</Window>
```

Malheureusement, ce n'est pas tout à fait aussi simple. Il nous faut encore démarrer le service permettant de réaliser les annotations. Cette partie se fait dans le code .NET.

Les membres suivants doivent être définis.

```
Private annotService As AnnotationService
Private annotStream As FileStream
Private annotStore As XmlStreamStore

Private Sub StartAnnotations()
   annotService = New AnnotationService(docViewer)
   annotStream = New FileStream("annotations.xml"_
                                , FileMode.OpenOrCreate _
                                , FileAccess.ReadWrite)
```

```
    annotStore = New XmlStreamStore(annotStream)
    annotService.Enable(annotStore)
End Sub
```

Dans cette méthode, le service assurant la gestion des notes est démarré et le fichier XML devant contenir les notes lui est associé. La méthode `StartAnnotations` doit être exécutée avant de pouvoir utiliser les notes. Vous pouvez utiliser soit l'événement `Loaded` pour le démarrer, soit le constructeur.

```
Public Sub New()
    InitializeComponent()
    StartAnnotations()
End Sub
```

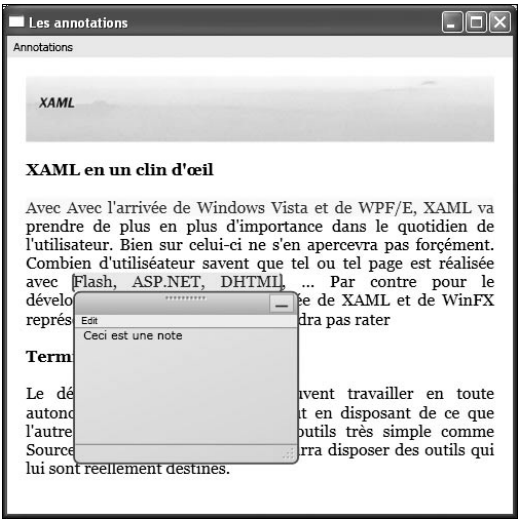
L'événement `Closed` a également été défini car, avant de quitter, il est nécessaire de fermer le fichier et le service utilisé par les annotations.

```
Private Sub StopAnnotations()
    annotStore.Flush()
    annotStream.Flush()
    annotStream.Close()
    annotStore = Nothing
End Sub
```

Maintenant, nous pouvons voir les résultats.

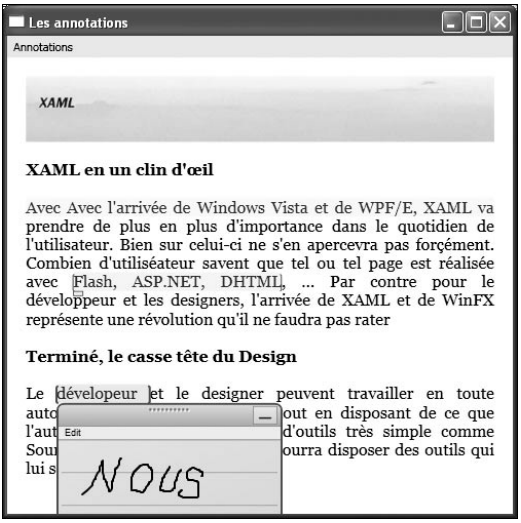
▲ **Figure 9-31** : Le menu des annotations

Comme vous pouvez le voir, le menu est désactivé. En effet, il est nécessaire de sélectionner du contenu pour pouvoir lui associer une note.



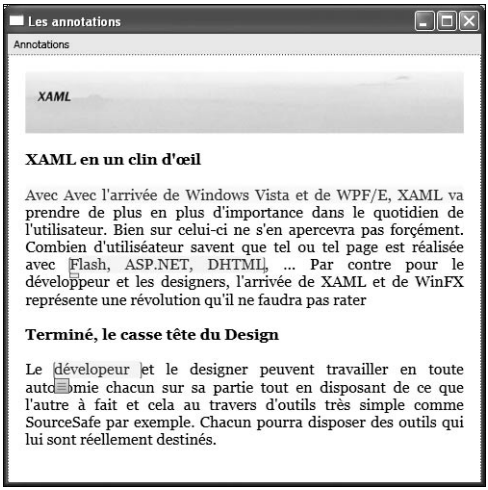
▲ **Figure 9-32 :** Surlignement et annotation

Dans cette seconde figure, vous devez pouvoir voir la première phrase qui a été surlignée et une note ouverte associée à une autre partie du texte.



▲ **Figure 9-33 :** Une note manuscrite

L'utilité de ce genre de note reste à démontrer car il n'est pas toujours facile de dessiner avec la souris.



▲ **Figure 9-34 :** Les notes fermées

La position des notes est conservée dans le fichier.

Attention

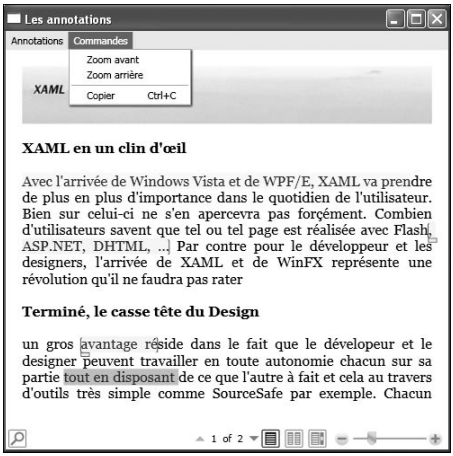
Modification du contenu

Si le contenu du document change, les notes ne seront plus associées au texte d'origine.

Nous avions déjà vu les commandes d'édition ; maintenant, nous venons de voir qu'il existe également des commandes prédéfinies pour les notes. En fait, il en existe bien d'autres dont les commandes d'application, de navigation, de composant…

À titre d'exemple, nous pouvons introduire l'une ou l'autre de ces commandes dans le code ci-dessus.

```
<MenuItem Header="Commandes" >
   <MenuItem Command="NavigationCommands.IncreaseZoom"
    Header="Zoom avant" />
   <MenuItem Command="NavigationCommands.DecreaseZoom"
    Header="Zoom arrière" />
   <Separator />
   <MenuItem Command="ApplicationCommands.Copy" Header="Copier" />
</MenuItem>
```



▲ **Figure 9-35 :** D'autres commandes

Remarque

Modification de la présentation

Si vous comparez le bas de cette illustration par rapport aux précédentes, vous constaterez l'apparition de la barre d'outils ! Cela est dû au fait que, pour ce dernier exemple, le `StackPanel` qui encadre l'ensemble a été remplacé par un `DockPanel`.

9.5 Checklist

Dans ce chapitre, les notions essentielles que nous avons vues sont :

- afficher un document pour que sa présentation reste inchangée ;
- afficher un document pour qu'il s'adapte au mieux à l'écran ;
- comment utiliser les contrôles spécifiques à la gestion du document comme les paragraphes, les sections, les tableaux, les listes à puce, les figures, les layers et les hyperliens ;
- comment utiliser un `BulletDecorator` pour réaliser des listes originales ;
- comment éditer et imprimer un document ;
- comment placer des annotations dans un document ;
- utiliser une barre de statut ;
- utiliser les commandes prédéfinies.

Les outils graphiques

Le designer de Visual Studio (nom de code CIDER) 290
Dans la gamme expression 303
Aurora Designer 310
ZAM 3D 313
Checklist 314

Bien que, comme nous l'avons vu, il soit possible et même facile de créer des interfaces graphiques même complexes en introduisant vous-même le code XAML nécessaire, les éditeurs ne vont pas manquer de vous proposer des outils graphiques pour réaliser vos écrans, vos pages ou toutes autres réalisations en XAML. De son côté, Microsoft prépare d'ores et déjà divers outils. Le premier sera intégré dans Visual Studio, les autres font partie de la gamme Expression. Microsoft n'est pas le seul acteur du marché actif dans le domaine. La société Mobiform a elle aussi un outil nommé Aurora Designer et destiné à la création d'interfaces utilisateurs XAML. Pour la création 3D, il existe l'outil ZAM 3D d'Electric Rain. Un autre acteur historique concernant XAML est XAMLON. Toutefois, au moment d'écrire ces lignes, la position du produit n'est pas très claire, que ce soit en termes de compatibilité ou de futur suite à l'intégration de la technologie Flash.

Attention

Il y a XAML et XAML

Outre le XAML de Microsoft, d'autres projets similaires tel *MyXAML* ont vu ou verront le jour. Ils offriront peut-être des outils graphiques qui leur sont dédiés. Ces projets ne sont pas forcément compatibles avec le XAML Microsoft, soyez prudent et assurez-vous que le produit respecte entièrement la syntaxe XAML. À moins bien sûr que vous n'optiez volontairement pour ce langage proche.

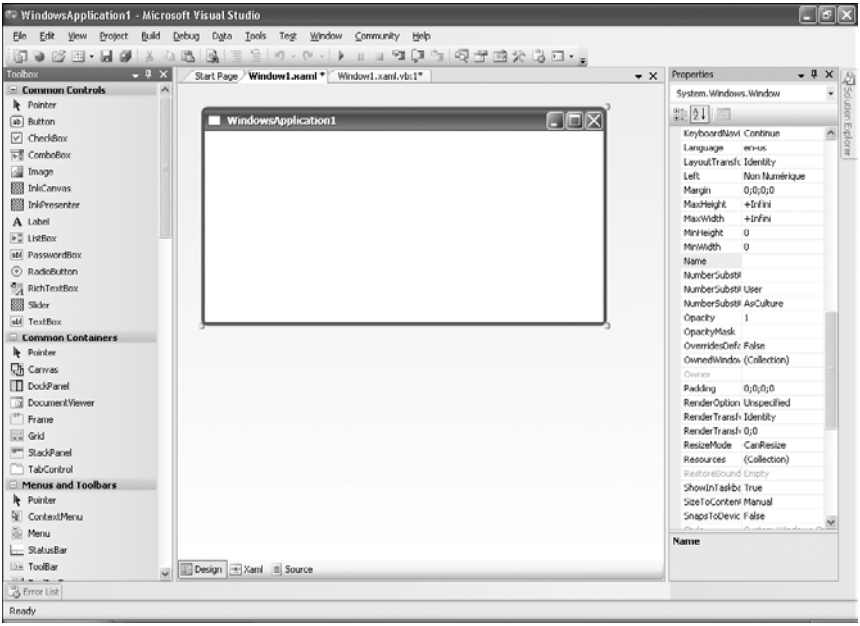
Dans les chapitres suivants, nous allons passer en revue ces différents produits. Il s'agit non pas d'apprendre à les manipuler mais uniquement de les découvrir sommairement et d'en comprendre les objectifs. Notez que tous ces produits sont actuellement en version bêta et par conséquent encore susceptibles d'être largement modifiés. Cette liste ne se veut pas exhaustive, et beaucoup d'autres logiciels auront probablement vu le jour ou ajouté le support du format XAML à leur propre logiciel.

10.1 Le designer de Visual Studio (nom de code CIDER)

Le premier de ces outils porte le nom de code de CIDER et est l'équivalent XAML du designer actuel inclus dans Visual Studio. Il s'y intègre d'ailleurs lui-même. L'appel à l'un ou l'autre designer est automatique en fonction du fichier que vous ouvrez. CIDER est inclus dans l'extension WinFX pour Visual Studio que vous avez vraisemblablement déjà installée. Il fera partie intégrante des prochaines versions de Visual Studio. C'est pourquoi nous nous y attarderons un peu plus.

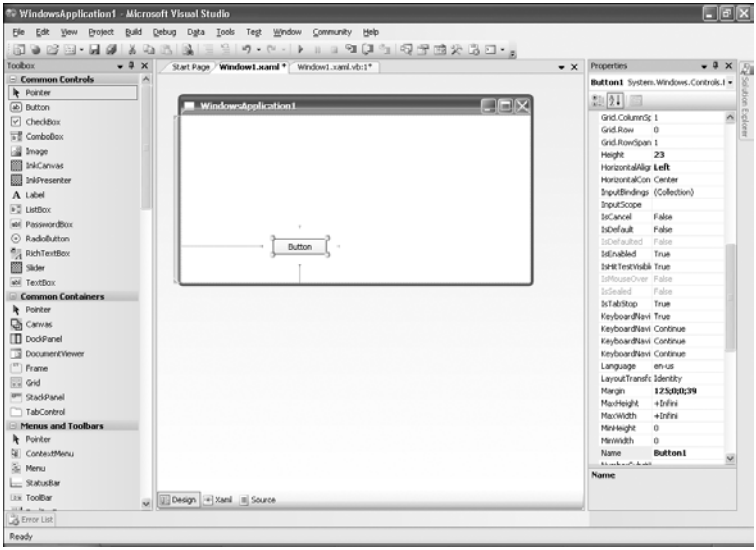
Comme vous pouvez le voir dans l'image ci-dessous, lorsque vous ouvrez le fichier XAML l'éditeur de Visual Studio vous permet de naviguer entre la vue en mode Design, la vue en mode XAML et le code source également appelé *code behind* (code associé se trouvant dans le fichier extension *.vb* ou *.cs*). Pour passer de l'un à l'autre, il suffit de cliquer sur l'onglet adéquat en bas à gauche de votre écran.

Le designer Cider ne diffère guère de votre designer Visual Studio habituel. Il présente au centre le résultat que vous allez obtenir.



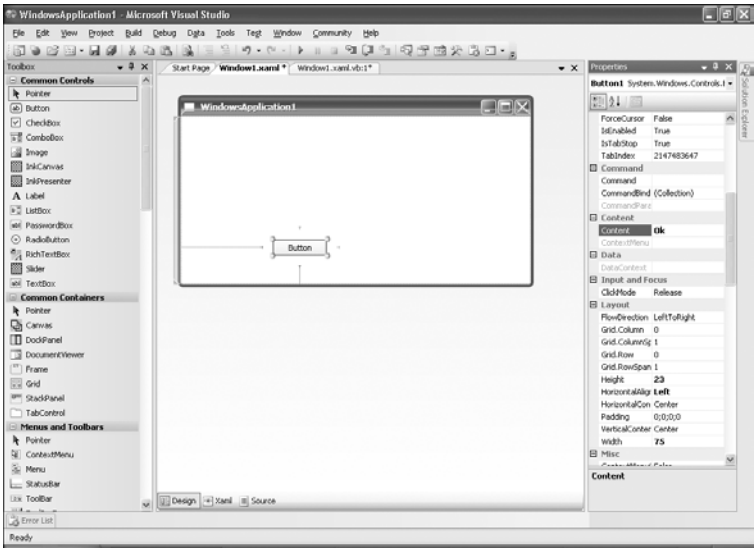
▲ **Figure 10-1 :** Design d'une Window

À gauche, vous retrouvez les composants que vous pouvez inclure. Pour ajouter un composant, il suffit de le sélectionner dans votre fenêtre de gauche et de le faire glisser où vous désirez le placer. Vous pouvez déplacer ou redimensionner les objets déjà placés dans la fenêtre centrale comme bon vous semble.



▲ **Figure 10-2 :** Ajout d'un bouton

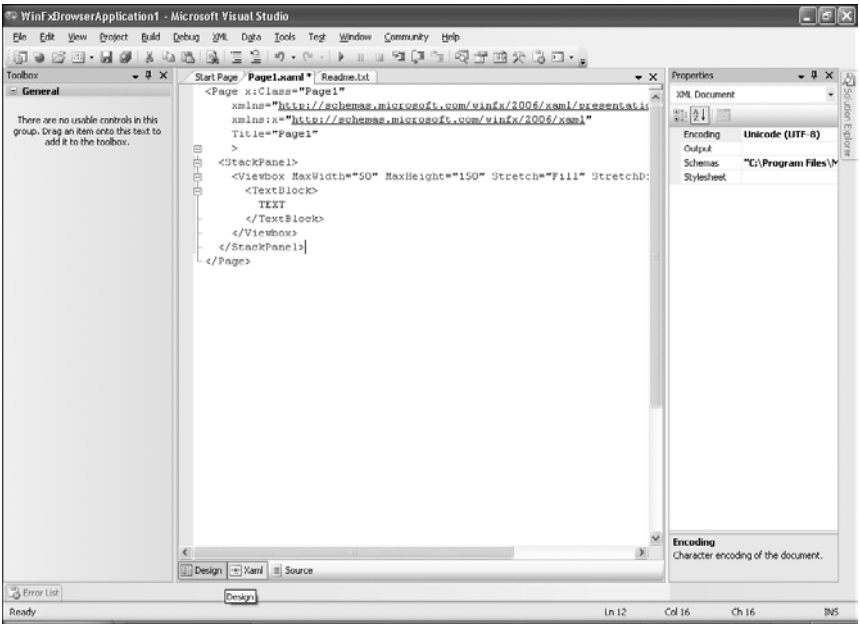
À droite, vous trouverez les propriétés de l'objet sélectionné, que vous pourrez adapter selon vos besoins.



▲ **Figure 10-3 :** Les propriétés

Bien sûr, comme pour toute l'interface Visual Studio, vous pouvez changer la disposition des différentes fenêtres.

Si vous désirez accéder directement au code ou simplement le visualiser, vous cliquez sur l'onglet **XAML**. Les modifications apportées au code sont directement répercutées dans l'affichage en mode Design.



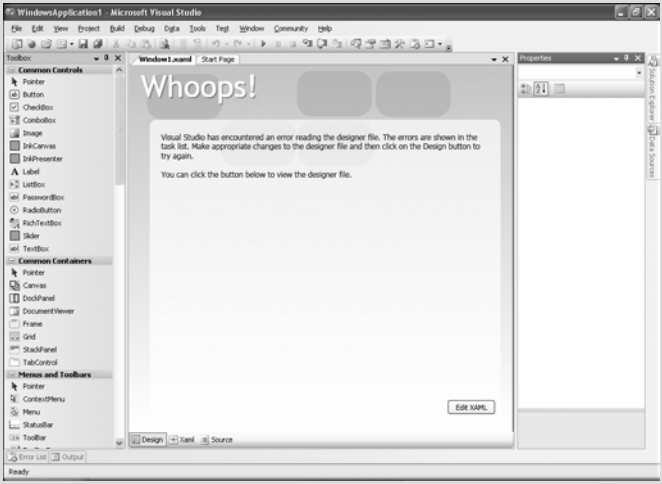
▲ **Figure 10-4 :** Vue du code

Attention

Code non valide

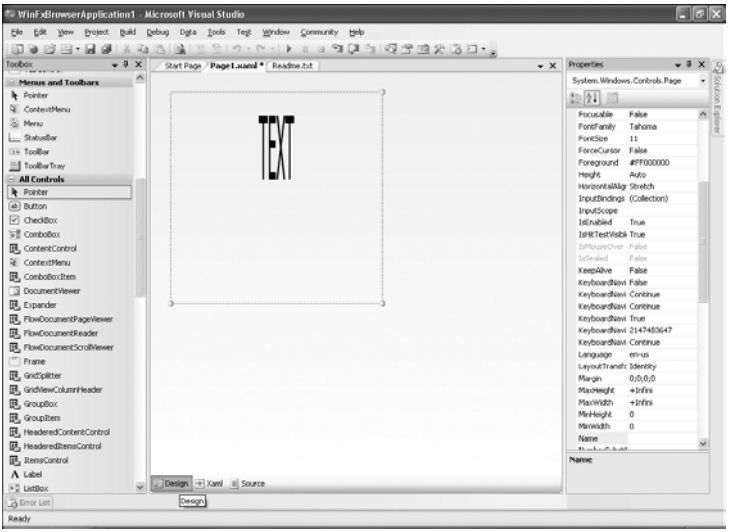
Si votre code est erroné, le mode Design ne pourra être activé.

Attention



▲ **Figure 10-5 :** Une erreur dans le code

Si votre projet concerne non pas une application Windows mais une page web, le designer fonctionne à l'identique excepté que la fenêtre Windows est remplacée par un cadre représentant les limites de la page.



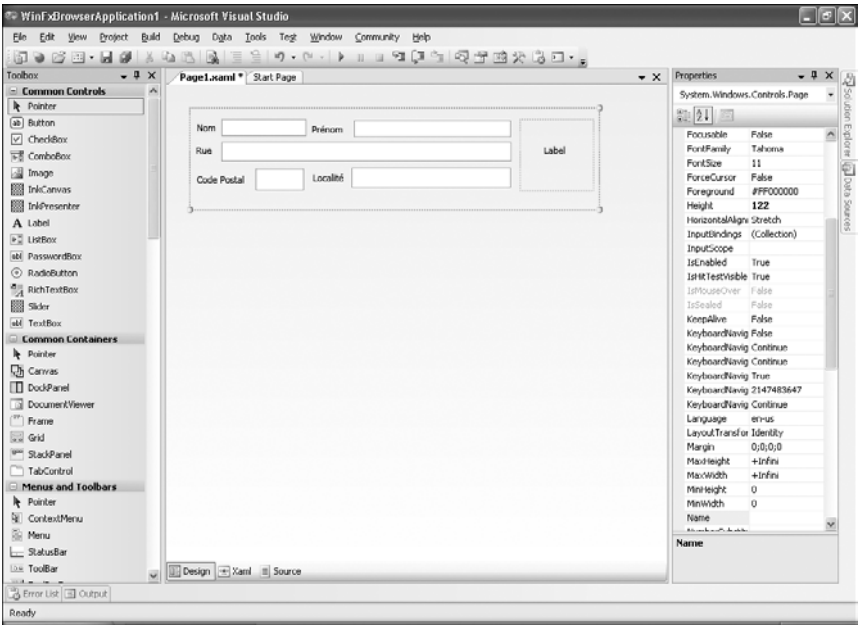
▲ **Figure 10-6 :** Design d'une page web

Nous pouvons reproduire l'exemple présenté dans le chapitre 55 mais cette fois réalisé non pas à la main mais avec le designer. Faites glisser les différents composants et n'oubliez pas de changer les propriétés `Name`, `MaxLength`...

Attention

`Label` au lieu de `TextBlock` !

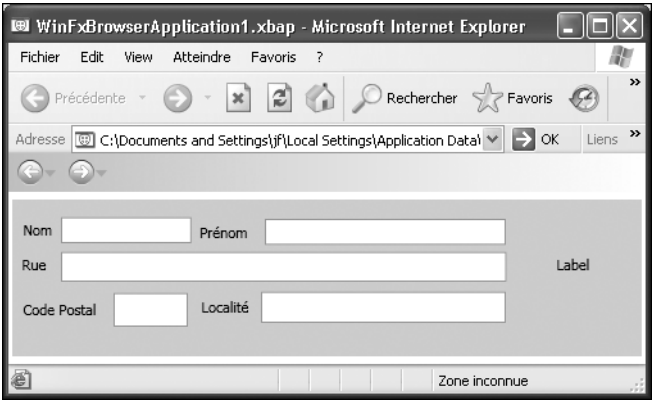
Tous les contrôles ne sont pas présents dans la barre d'outils. Pour l'exemple, choisissez un contrôle similaire qui vous permettra d'obtenir un résultat proche.



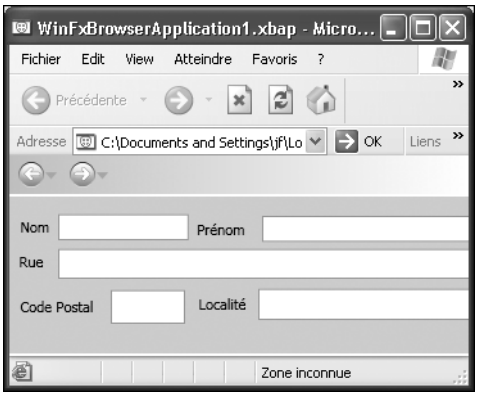
▲ **Figure 10-7 :** Design d'une page simple

Lors de l'exécution de notre page, nous obtenons un résultat fort similaire (voir Figure 10-8).

Lorsque nous redimensionnons la page, le comportement est semblable à celui obtenu avec notre balise `Canvas` (voir Figure 10-9).



▲ **Figure 10-8 :** La page affichée



◄ **Figure 10-9 :** La page redimensionnée

Et, pourtant, si nous observons le code généré, nous nous apercevrons qu'il s'agit d'une grille.

```
<Page x:Class="Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page1" Height="122" Width="491">
    <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="0.4*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.882470119521912*" />
    </Grid.ColumnDefinitions>
```

```
    <Label HorizontalAlignment="Left" Margin="3.37,11.7233333333333,0,0"
      Name="lblNom" Width="36.63" Height="25.2766666666668"
      VerticalAlignment="Top">Nom</Label>
    <TextBox Margin="37.9999999999999,13.7233333333333,0,0" Name="txtNom"
      CharacterCasing="Upper" MaxLength="30" HorizontalAlignment="Left"
      Width="101.709090909091" Height="20.2766666666668"
      VerticalAlignment="Top"></TextBox>
    <Label Margin="141.079090909091,13.7233333333333,0,0" Name="lblPrenom"
      Height="23.2766666666668" VerticalAlignment="Top"
      HorizontalAlignment="Left" Width="54.211818181818">Prénom</Label>
    <TextBox Margin="197,15,106,0" Name="txtPrenom" MaxLength="30"
      Height="20.2766666666668" VerticalAlignment="Top" ></TextBox>
    <Label HorizontalAlignment="Left" Margin="2.37,38.7233333333333,0,58"
      Name="lblAdr" Width="35.63">Rue</Label>
    <TextBox Margin="38,40.7233333333333,105.709090909091,58" MaxLength="80"
      Name="txtAdr" ></TextBox>
    <Label Height="21.2766666666667" HorizontalAlignment="Left"
      Margin="3.37,0,0,27.0000000000001" Name="lblCP"
      VerticalAlignment="Bottom" Width="70.63">Code Postal</Label>
    <TextBox Height="26.2766666666667" HorizontalAlignment="Left"
      Margin="79,0,0,23" MaxLength="5" Name="txtCP"
      VerticalAlignment="Bottom" Width="58"></TextBox>
    <Label Height="22.2766666666667" Margin="142.37,0,103,28"
      Name="lblLocalite" VerticalAlignment="Bottom">Localité</Label>
    <TextBox Height="24.2766666666667" Margin="194,0,106,26" MaxLength="50"
      Name="txtLocalite" VerticalAlignment="Bottom"></TextBox>
    <Canvas HorizontalAlignment="Right" Margin="0,14,7,23" MinHeight="50"
      MinWidth="50" Name="Canvas1" Width="88" />
    <Label HorizontalAlignment="Right" Margin="0,38.7233333333333,36,58"
      Name="blkPhoto" Width="35.63">Label</Label>
  </Grid>
</Page>
```

Remarque

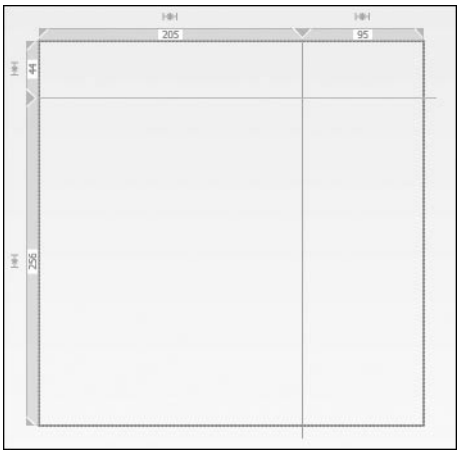
Lisibilité du code

Le code vous est présenté exactement comme il est généré. C'est-à-dire une ligne de code par contrôle. La lisibilité du code est loin d'être assurée.

Comme vous pouvez le constater, le code est loin d'être propre et il vous faudra le revoir vous-même ou du moins adapter *via* la fenêtre des propriétés certains éléments pour obtenir des valeurs plus adéquates. Évidemment, vous pouvez laisser tout ainsi et décider de toujours utiliser le designer. Dans ce cas, espérons seulement que vous n'ayez jamais à être contraint, pour des raisons techniques, à devoir malgré tout accéder directement au code.

Donc, à la lecture du code, nous avons une grille d'une ligne et d'une colonne. Les éléments sont placés dans cette cellule selon leur ordre et en jouant sur les tailles et les marges de chacun des éléments.

Il est évidemment possible de créer des lignes et des colonnes avec le designer. Il suffit pour cela de sélectionner la ligne à gauche, petit triangle en haut à gauche, ou la ligne en bas à droite, même triangle, et de le faire glisser là où vous le désirez. Vous pouvez aussi déplacer une de ces lignes pour changer la taille d'une ligne ou d'une colonne de la grille.

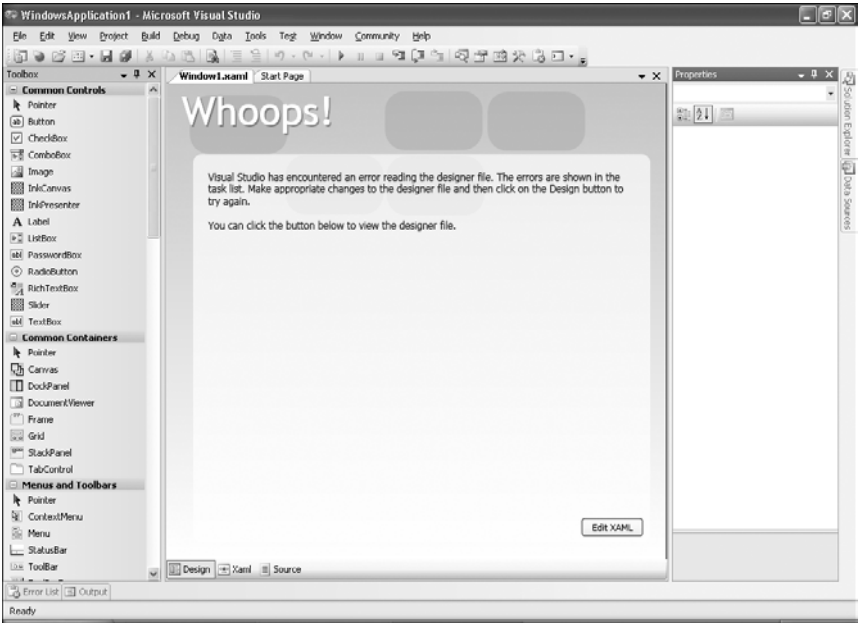


◄ **Figure 10-10 :** Les repères de la grille

Ces problèmes éventuels de lisibilité sont l'occasion de parler un peu du débogage d'une application XAML. En réalité, il y a débogage non pas à proprement parler du code XAML mais seulement du code .NET qui lui est associé. C'est donc un autre motif pour essayer d'avoir un code propre et clair. Comme le code dépend peu d'une situation particulière, l'exception générée en cas d'erreur suffit largement pour comprendre et résoudre le problème. En revanche, il est toutefois possible de déboguer du code .NET que vous avez écrit indirectement grâce au XAML.

Renvoi *Pour plus d'information sur ce code, reportez-vous au chapitre* ***Utiliser une grille*** *page 61.*

Pour accéder facilement à ce code avec le débugger, vous ouvrez la source .NET de la page ou de la fenêtre à laquelle vous désirez accéder. Vous placez un point d'arrêt sur la ligne `InitializeComponent()`, que vous trouverez dans le constructeur. Pour rappel, pour placer facilement un point d'arrêt vous placez le curseur sur la ligne voulue et vous appuyez sur la touche F9.



▲ **Figure 10-11** : Le mode debug

Vous demandez ensuite l'exécution du programme en mode debug en utilisant la touche F5. Il vous reste alors à manipuler le programme pour accéder à la page ou à l'écran voulu. À ce moment, le programme s'arrête. Appuyez alors classiquement sur la touche F11 pour poursuivre à l'intérieur de la procédure appelée et vous êtes en débogage dans le fichier *g.vb* correspondant. À titre d'exemple, vous trouverez ci-dessous le fichier correspondant à notre exemple. À vous alors de poursuivre le débogage.

```
'------------------------------------------------------------
' <auto-generated>
'     This code was generated by a tool.
'     Runtime Version:2.0.50727.42
'
'     Changes to this file may cause incorrect behavior
'     and will be lost if the code is regenerated.
' </auto-generated>
'------------------------------------------------------------

Option Strict Off
Option Explicit On

Imports System
```

```
Imports System.Windows
Imports System.Windows.Automation
Imports System.Windows.Controls
Imports System.Windows.Controls.Primitives
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Input
Imports System.Windows.Markup
Imports System.Windows.Media
Imports System.Windows.Media.Animation
Imports System.Windows.Media.Effects
Imports System.Windows.Media.Imaging
Imports System.Windows.Media.Media3D
Imports System.Windows.Media.TextFormatting
Imports System.Windows.Navigation
Imports System.Windows.Shapes


'''<summary>
'''Page1
'''</summary>
<
Microsoft.VisualBasic.CompilerServices.DesignerGenerated()
> _
Partial Public Class Page1
    Inherits System.Windows.Controls.Page
    Implements System.Windows.Markup.IComponentConnector

    Friend WithEvents lblNom As System.Windows.Controls.Label

    Friend WithEvents txtNom As System.Windows.Controls.TextBox

    Friend WithEvents lblPrenom As System.Windows.Controls.Label

    Friend WithEvents txtPrenom As System.Windows.Controls.TextBox

    Friend WithEvents lblAdr As System.Windows.Controls.Label

    Friend WithEvents txtAdr As System.Windows.Controls.TextBox

    Friend WithEvents lblCP As System.Windows.Controls.Label

    Friend WithEvents txtCP As System.Windows.Controls.TextBox

    Friend WithEvents lblLocalite As System.Windows.Controls.Label

    Friend WithEvents txtLocalite As System.Windows.Controls.TextBox

    Friend WithEvents Canvas1 As System.Windows.Controls.Canvas
```

```
    Friend WithEvents blkPhoto As System.Windows.Controls.Label

    Private _contentLoaded As Boolean

    '''<summary>
    '''InitializeComponent
    '''</summary>
    Public Sub InitializeComponent()
        Implements System.Windows.Markup.
             IComponentConnector.InitializeComponent
        If _contentLoaded Then
            Return
        End If
        _contentLoaded = true
        Dim resourceLocater As System.Uri = _
          New System.Uri( _
          "WinFxBrowserApplication1;component\page1.baml" _
                    , System.UriKind.RelativeOrAbsolute)
        System.Windows.Application.LoadComponent(Me _
                                        , resourceLocater)
    End Sub

    Sub
System_Windows_Markup_IComponentConnector_Connect( _
                          ByVal connectionId As Integer _
                        , ByVal target As Object) _
  Implements _
          System.Windows.Markup.IComponentConnector.Connect
        If (connectionId = 1) Then
            Me.lblNom = _
                CType(target,System.Windows.Controls.Label)
            Return
        End If
        If (connectionId = 2) Then
            Me.txtNom = _
              CType(target,System.Windows.Controls.TextBox)
            Return
        End If
        If (connectionId = 3) Then
            Me.lblPrenom = _
              CType(target,System.Windows.Controls.Label)
            Return
        End If
        If (connectionId = 4) Then
            Me.txtPrenom = _
              CType(target,System.Windows.Controls.TextBox)
            Return
        End If
        If (connectionId = 5) Then
            Me.lblAdr = _
```

```
                CType(target,System.Windows.Controls.Label)
            Return
        End If
        If (connectionId = 6) Then
            Me.txtAdr = _
              CType(target,System.Windows.Controls.TextBox)
            Return
        End If
        If (connectionId = 7) Then
            Me.lblCP = _
               CType(target,System.Windows.Controls.Label)
            Return
        End If
        If (connectionId = 8) Then
            Me.txtCP = _
              CType(target,System.Windows.Controls.TextBox)
            Return
        End If
        If (connectionId = 9) Then
            Me.lblLocalite = _
                CType(target,System.Windows.Controls.Label)
            Return
        End If
        If (connectionId = 10) Then
            Me.txtLocalite = _
              CType(target,System.Windows.Controls.TextBox)
            Return
        End If
        If (connectionId = 11) Then
            Me.Canvas1 = _
               CType(target,System.Windows.Controls.Canvas)
            Return
        End If
        If (connectionId = 12) Then
            Me.blkPhoto = _
               CType(target,System.Windows.Controls.Label)
            Return
        End If
        Me._contentLoaded = true
    End Sub
End Class
```

Lorsqu'il y a une faute au cours de l'exécution dans la partie du programme écrite en XAML, c'est ce code que le debugger va vous montrer.

10.2 Dans la gamme expression

La gamme expression est une nouvelle gamme de logiciels Microsoft destinée au designer. Elle se compose de trois programmes distincts : Graphic Designer pour le graphisme, Interactive Designer pour le design d'application XAML et Web Designer pour le développement de page web.

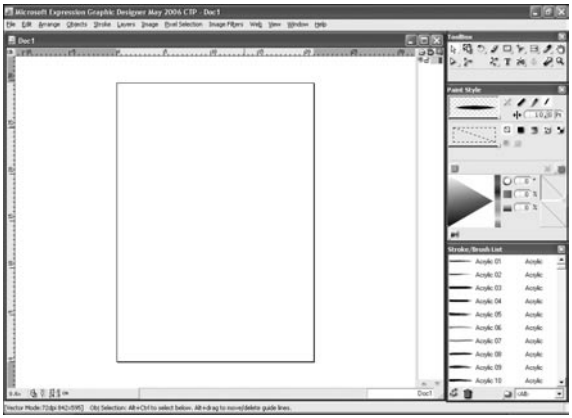
Graphic Designer

◄ **Figure 10-12 :** Expression Graphic Designer

Comme son nom l'indique, il s'agit d'un outil de design destiné à celui qui souhaite réaliser du graphisme. Il ne s'agit pas *a priori* d'un pur outil XAML puisqu'il travaille avec un autre format de fichier. En revanche, il dispose d'un outil d'exportation vers le XAML, ce qui fait de lui un candidat idéal pour réaliser les graphismes que vous souhaitez introduire dans vos développements.

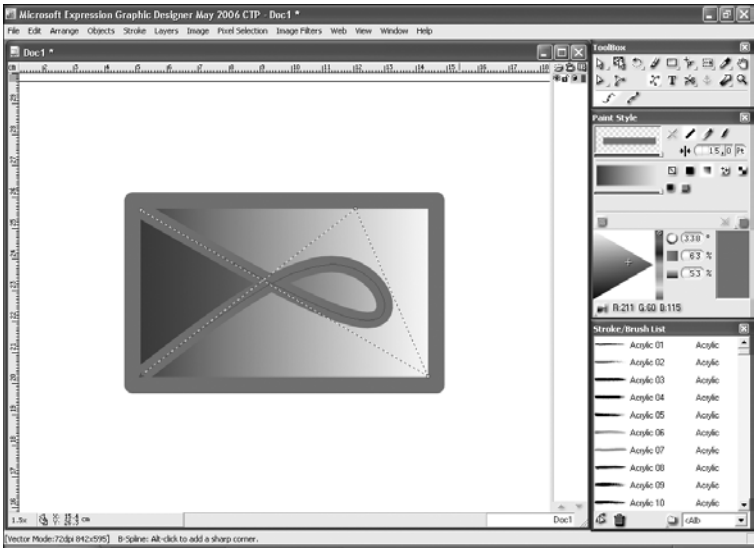
Le but de ce livre n'étant clairement pas de vous expliquer le fonctionnement de cet outil, qui mérite à lui seul un livre, vous ne trouverez dans ce chapitre qu'une simple présentation de l'outil sans même entrer dans ses possibilités.

La page principale de Graphic Designer est très classique avec la page proprement dite à droite et les fenêtres d'outils présentées par défaut à gauche.



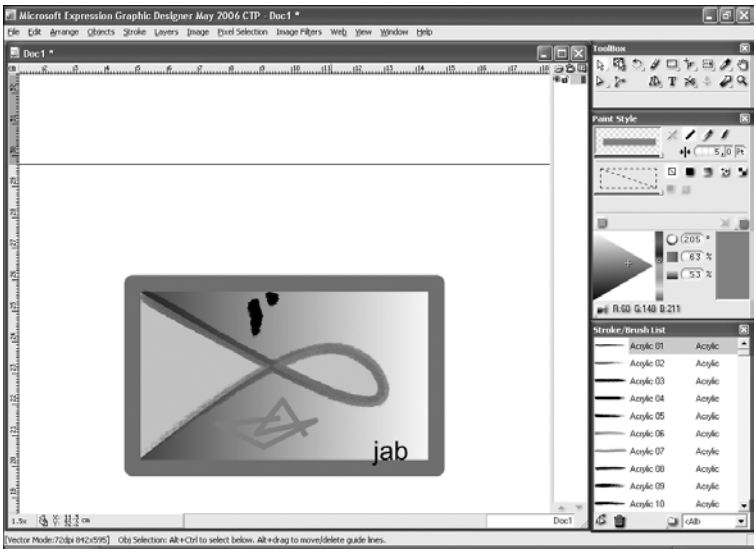
▲ **Figure 10-13 :** La page principale

Le traçage des objets se fait classiquement en utilisant des points de référence.



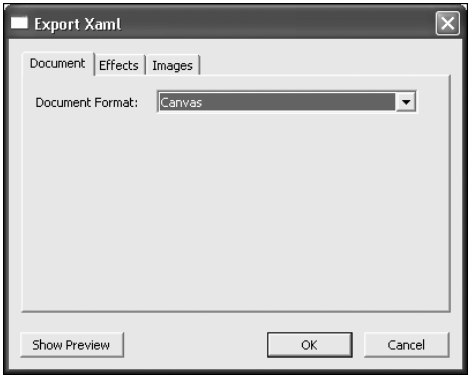
▲ **Figure 10-14 :** Tracer une forme

Il est évidemment possible de changer après coup les motifs dessinés.



▲ **Figure 10-15 :** Modifier la forme

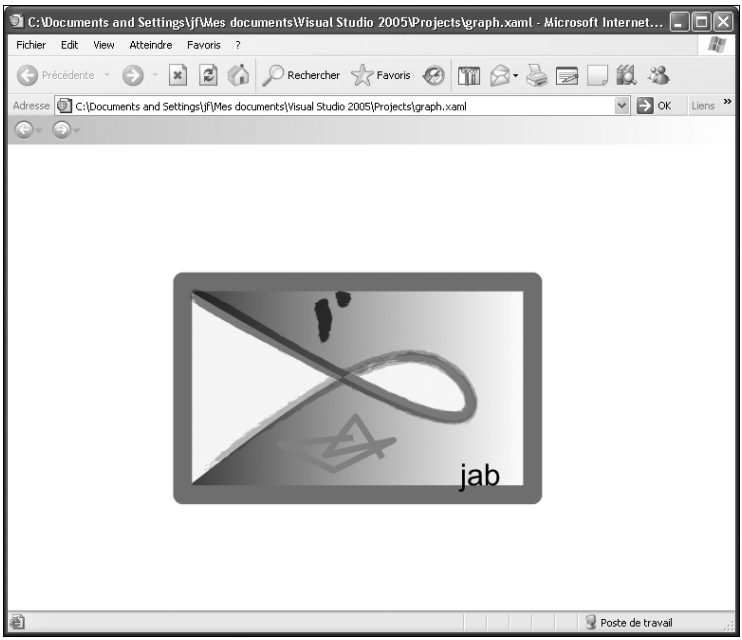
Pour exporter le résultat en XAML, il suffit d'utiliser la fonction d'exportation.



◄ **Figure 10-16 :** Exporter en XAML

Celle-ci vous permet de définir quelques paramètres comme l'utilisation d'un `Canvas`.

Une fois le fichier exporté, il ne vous reste qu'à l'incorporer dans votre programme ou à l'afficher directement dans un browser.



▲ **Figure 10-17 :** Le dessin dans un navigateur Web

Comme vous pouvez le constater, le rendu n'est pas totalement identique mais Graphic Designer est lui aussi en version bêta et nous pouvons espérer que ces problèmes soient prochainement résolus.

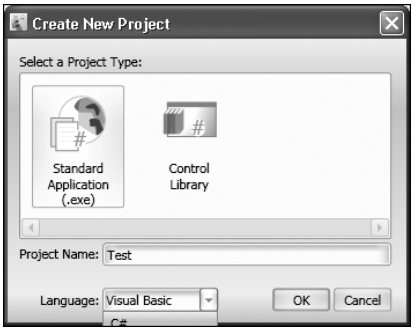
Vous pouvez également ouvrir le fichier XAML généré mais, comme nous pouvions nous y attendre, celui-ci est relativement touffu. Il contient plus de 400 nœuds pour un total de plus de 150 000 caractères.

Interactive Designer

◄ **Figure 10-18 :** Expression Interactive Designer

Interactive Designer est un outil qui peut être considéré soit comme un outil de conception complet, soit comme un outil complémentaire à Visual Studio et à Cider. Grâce à sa puissante interface utilisateur, il permet de créer des pages XAML très complètes et incluant des animations entièrement créées visuellement.

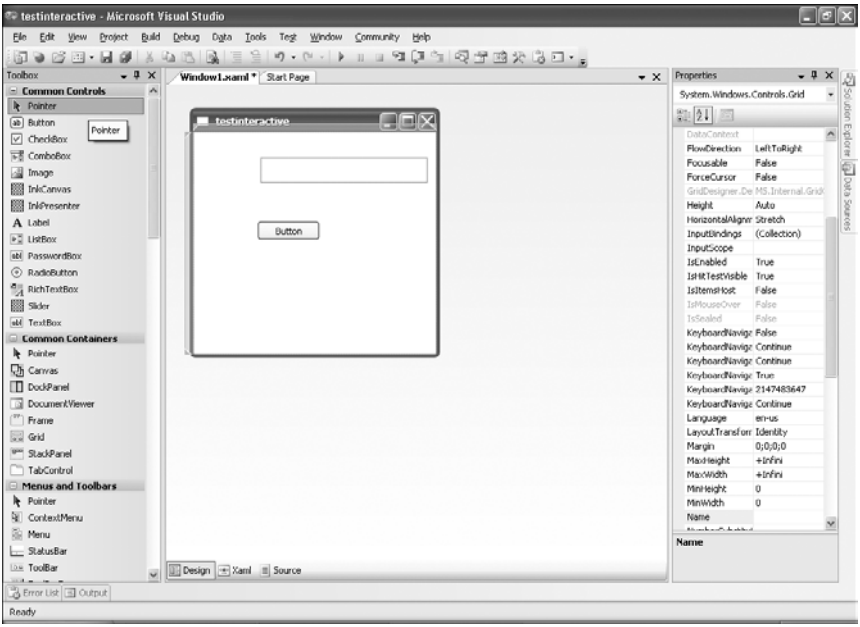
Comme Visual Studio, il travaille sur la base de projets contenant l'ensemble d'une application. Celle-ci sera alors compilée. Vous avez le choix entre le langage VB.NET ou C#. Avec ceux-ci, vous pourrez encoder du code .NET. Il est également capable de compiler et d'exécuter les projets. C'est pourquoi il peut être considéré comme un outil complet. Il n'offre toutefois pas les facilités de développement de code qu'offre Visual Studio. L'idéal étant d'utiliser les deux.



◄ **Figure 10-19 :** Choix du langage

Les fichiers projets sont compatibles avec ceux de Visual Studio, ce qui permet de passer facilement de l'un à l'autre et rend ces outils complémentaires, Visual Studio étant destiné au développeur et Interactive Designer, comme son nom l'indique, au designer.

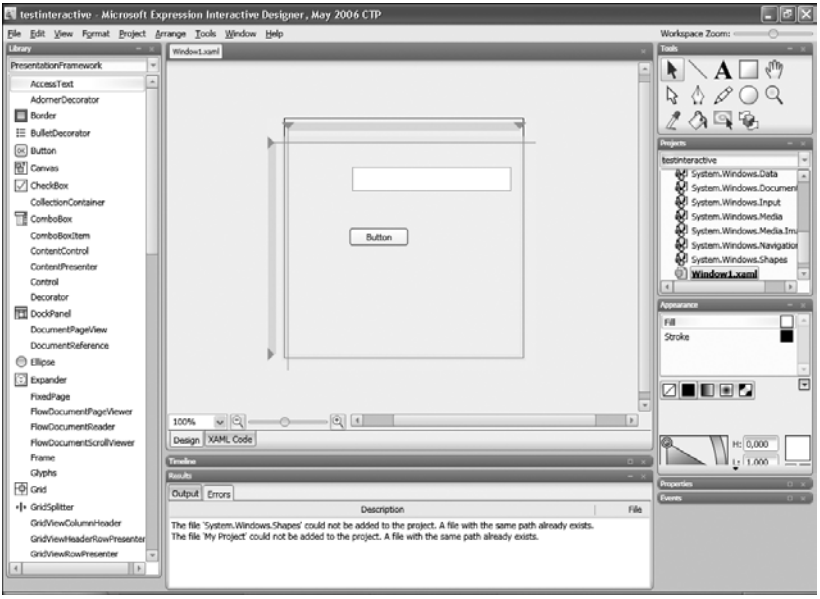
Créons avec Visual Studio un petit projet où le développeur n'a fait que placer les contrôles dont il avait besoin et sans se soucier de l'esthétique.



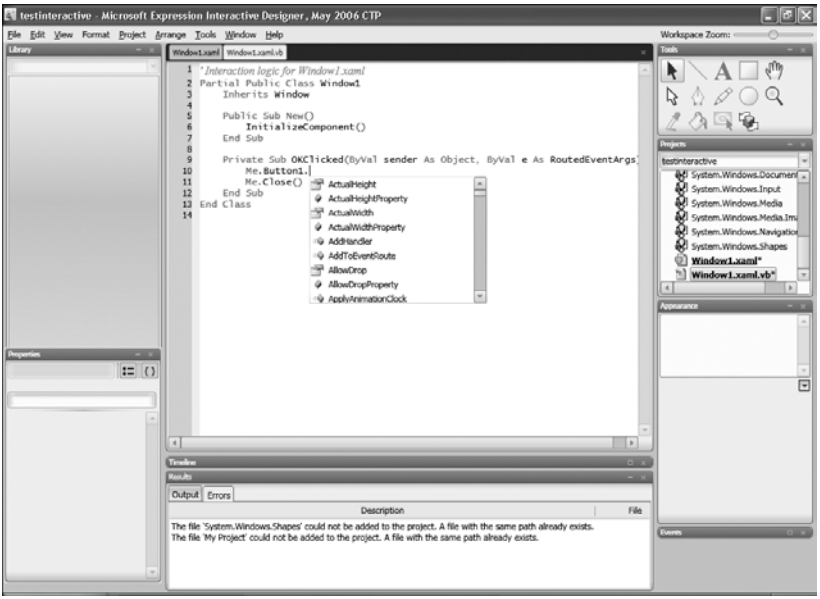
▲ **Figure 10-20 :** Un projet en Visual Studio

Récupérons-le dans Interactive Designer simplement en faisant **Open Project** (voir Figure 10-21).

Le projet est parfaitement récupéré et la fenêtre est affichée telle quelle. Le designer peut alors réaliser la mise en page. Si nécessaire, il peut avoir accès au code et même à l'IntelliSense (voir Figure 10-22).

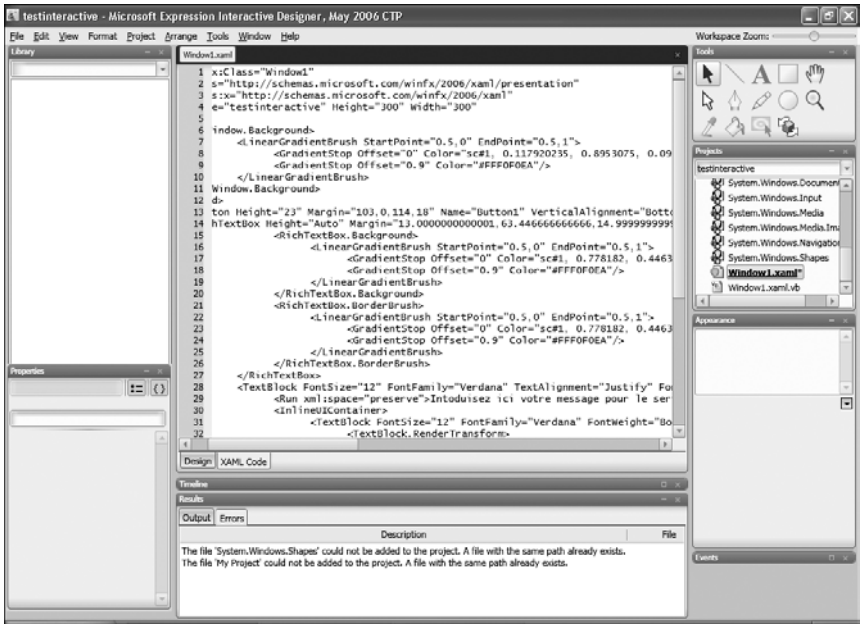


▲ **Figure 10-21 :** Un projet en Visual Studio



▲ **Figure 10-22 :** Le code .NET dans Interactive Designer

Il peut évidemment aussi accéder au code XAML.

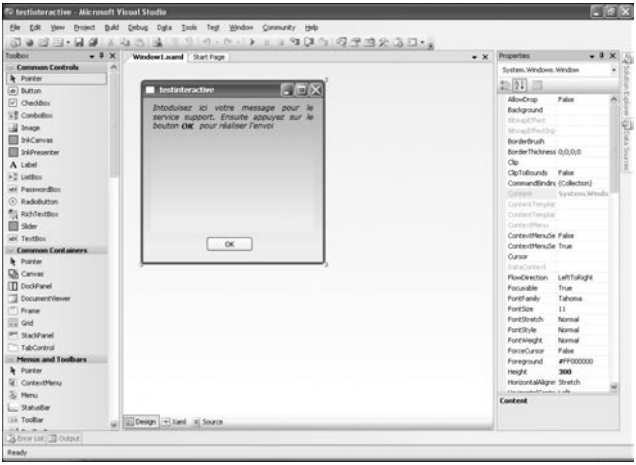


▲ **Figure 10-23 :** Le code XAML dans Interactive Designer

Une fois les modifications apportées, il peut exécuter l'application pour voir le résultat.

◄ **Figure 10-24 :** Exécution depuis Interactive Designer

Le projet peut parfaitement être à nouveau ouvert dans Visual Studio, qui prendra parfaitement en compte les modifications apportées.



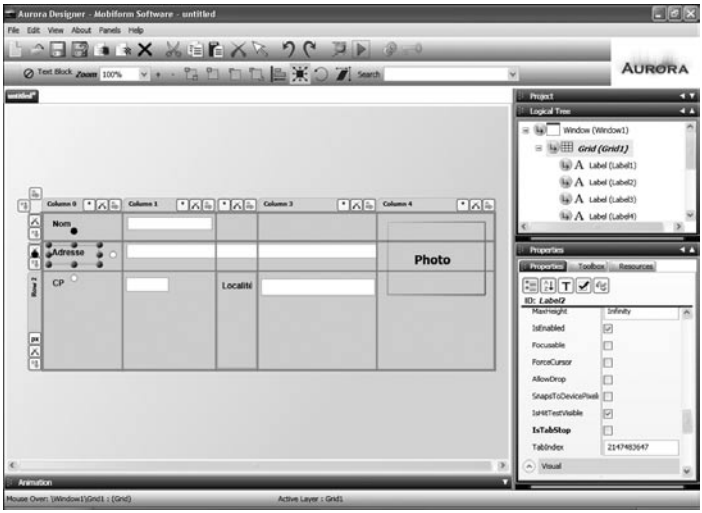
▲ **Figure 10-25 :** Projet réouvert avec Visual Studio

10.3 Aurora Designer

Aurora Designer est produit par la société Mobiform. Il est une alternative aux produits Microsoft et offre lui aussi un outil puissant de design XAML. Il offre également des composants supplémentaires pour enrichir encore les possibilités de XAML. En revanche, comme d'ailleurs Graphic Designer, il s'agit d'un outil exclusivement XAML sans support du code .NET qui devra être géré séparément si vous en avez besoin.

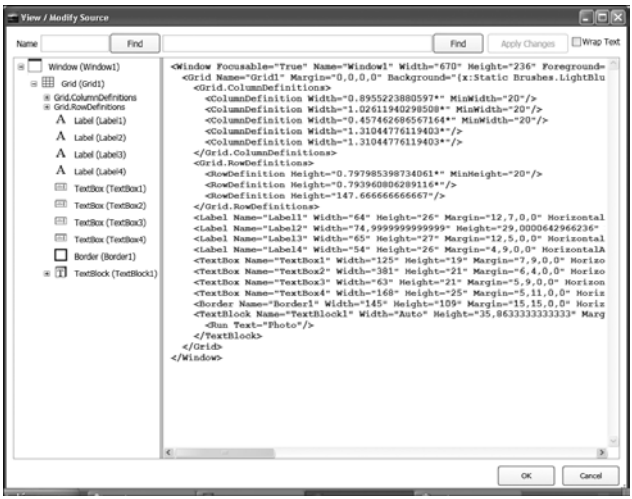
▲ **Figure 10-26 :** Choix du langage

L'interface d'Aurora est tout aussi classique que les précédentes avec une fenêtre de design et des fenêtres déplaçables pour la boîte à outils, les propriétés, les fichiers du projet et ainsi de suite.



▲ **Figure 10-27 :** L'interface d'Aurora

Selon le type de fichier choisi lors de la création, Aurora vous propose un conteneur adapté. Pour une fenêtre Windows, il s'agit par défaut d'une grille.



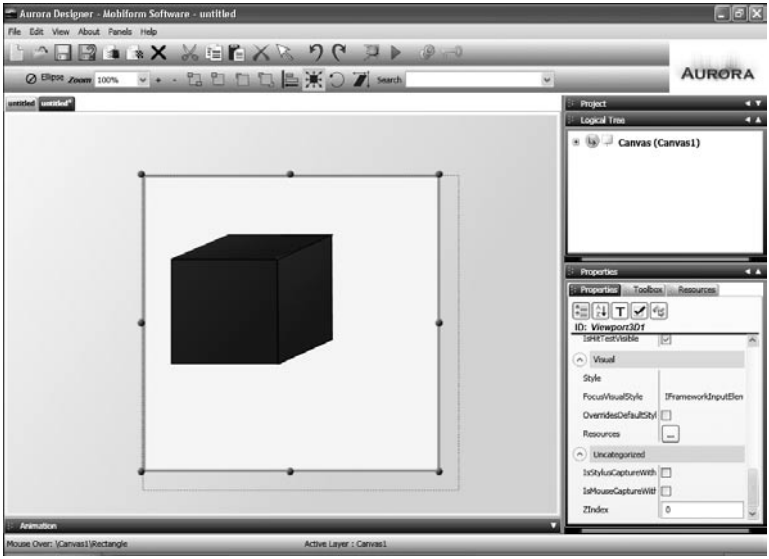
▲ **Figure 10-28 :** Design d'une fenêtre

Vous pouvez non seulement visualiser le code XAML mais également le modifier. La partie gauche de l'écran vous facilite la navigation.

Si nous transférons le code créé dans XamlPad, il s'affiche sans problème.

▲ **Figure 10-29 :** Le code dans XamlPad

Nous pouvons aussi créer des projets 3D mais aucun élément spécifique comme une sphère ou même un cube n'est prévu. Ce qui est proposé correspond au XAML de base uniquement. Il est malgré tout possible de réaliser rapidement un cube en utilisant trois rectangles et en leur appliquant des transformations de manière entièrement visuelle.

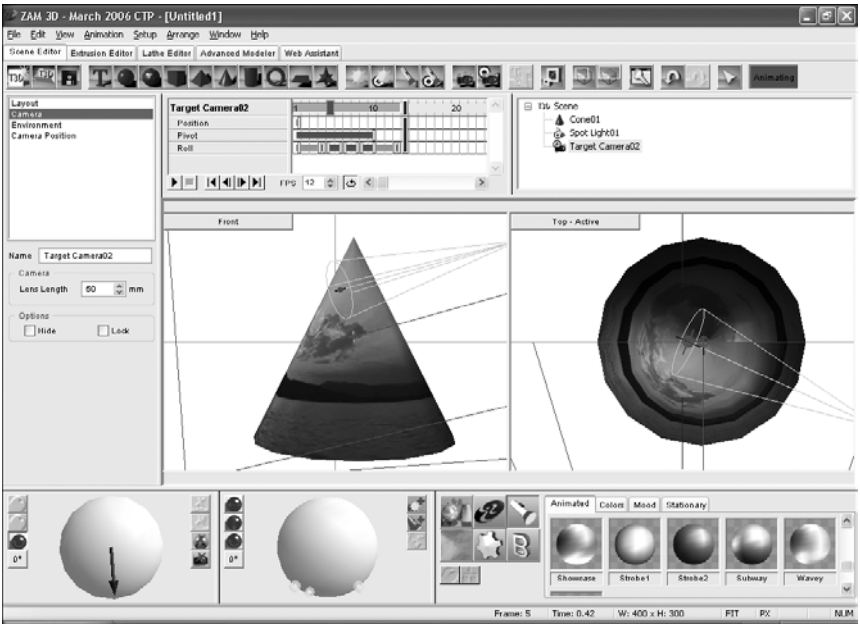


▲ **Figure 10-30 :** Un cube

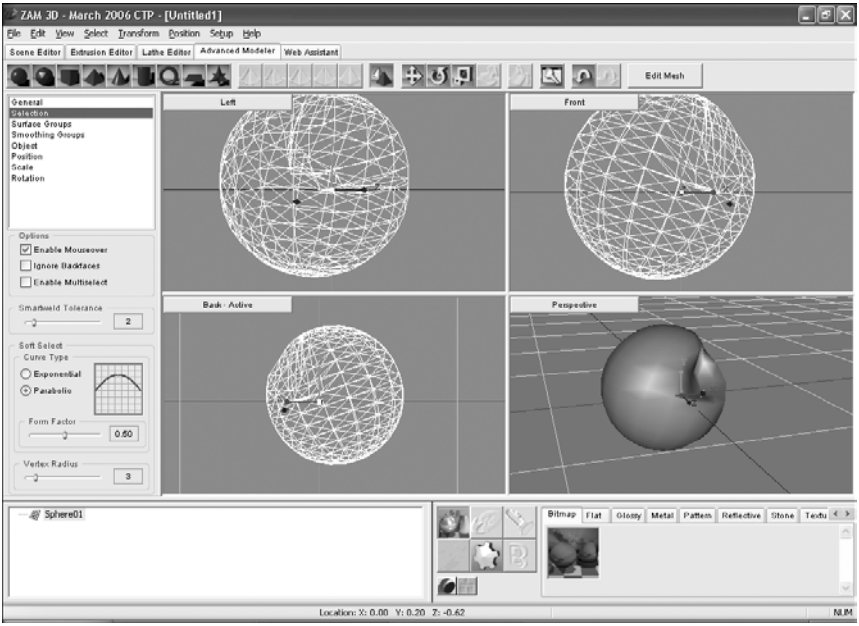
Malheureusement, le code créé pour le cube ne semblait pas compatible avec XamlPad ou Visual Studio. Ce genre de problème sera vraisemblablement corrigé pour la version définitive du produit. Il est en effet difficile pour un concepteur tiers de réaliser un logiciel destiné à un environnement qui est encore en mutation.

10.4 ZAM 3D

ZAM 3D, de la société Electric Rain, est un outil complètement orienté 3D. Il offre une panoplie de formes 3D prédéfinies ainsi que la création et le positionnement de spot et de caméra par simple *drag and drop*. Vous aurez également avec cet outil la possibilité de définir des animations grâce à la gestion des lignes de temps. Le résultat de votre travail peut être sauvé en format XAML soit sous forme d'une page de code soit sous forme d'une ressource que vous pourrez inclure dans vos développements.



▲ **Figure 10-31 :** L'interface de ZAM 3D



▲ **Figure 10-32 :** Une autre vue de ZAM 3D

10.5 Checklist

Dans ce chapitre, nous avons parcouru les outils les plus connus actuellement et nous avons vu comment ils peuvent ou ne peuvent pas nous aider. Les fonctionnalités essentielles sont :

- l'assistance au développement avec Visual Studio et son extension pour WinFX ;
- le travail de design avec Interactive Designer et Aurora Designer ;
- la composition 3D avec ZAM 3D ;
- le XAML dans le monde du graphisme avec Graphic Designer.

Le dessin

Le dessin en 2D 316
Le dessin en 3D 323
Checklist 327

Dans l'informatique moderne, le visuel a pris une très grande importance. C'est pourquoi, plutôt que limiter les possibilités de l'affichage à des images prédéfinies, XAML intègre des fonctionnalités de dessin relativement avancées. Contrairement aux fonctions de dessin souvent rencontrées jusqu'alors, il s'agit non pas de bitmap mais bien de vectoriel. Le langage offre en fait toutes les fonctions nécessaires à la création d'images vectorielles 2D ou 3D et à leur animation. Ce qui en fait une plate-forme ouverte également au monde de la CAO, par exemple, et va permettre assez facilement de mélanger du contenu très riche en terme de dessin avec du contenu plus classique.

11.1 Le dessin en 2D

XAML ne nous offre pas seulement les possibilités de réaliser des écrans composés de contrôles divers ; nous pouvons également réaliser des dessins. Il nous offre une gamme de classe relativement complète allant du dessin 2D à la 3D. Toutefois, pour bien dessiner, il faut avant tout savoir dessiner. L'objectif n'est pas ici de vous apprendre à dessiner mais il s'agit seulement de vous montrer quelques possibilités offertes par XAML avec des réalisations très simples.

Pour commencer, voyons tout d'abord comment tracer une ligne. Ce qui se fait naturellement avec la balise `Line`.

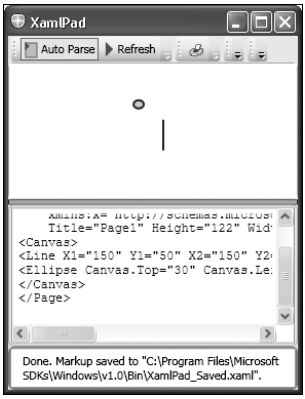
```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
   </Canvas>
</Page>
```

◄ **Figure 11-1 :** Un simple trait

Comme vous pouvez le constater, pour tracer une ligne vous devez définir le point d'origine aux attributs `X1` et `Y1` et le point de destination avec `X2` et `Y2`. L'origine des coordonnées correspond au coin supérieur gauche du conteneur, dans ce cas le `Canvas`. L'attribut `Stroke` fixe la couleur du trait alors que `StrokeThickness` en fixe la largeur. Ces deux derniers attributs sont obligatoires si vous désirez voir le résultat.

XAML dispose aussi de formes géométriques. L'ellipse est une de ces formes.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="120"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="165"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
   </Canvas>
</Page>
```



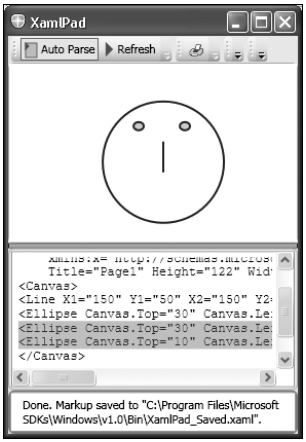
◄ **Figure 11-2 :** Une ellipse

La balise `Ellipse` trace une ellipse dont les dimensions sont fixées par les attributs `Height` et `Width`. Le positionnement de l'ellipse diffère de celui de la ligne mais est plus conforme à ce que nous avions utilisé jusqu'ici puisqu'il faut utiliser les propriétés attachées du conteneur; ici de `Canvas`.

La propriété `Fill` permet de remplir le fond d'une forme.

Pour tracer un cercle, il suffit de tracer une ellipse dont les deux dimensions sont égales. Partant de ce principe, XAML n'a pas intégré de classe `Cercle`.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="120"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Canvas.Top="10" Canvas.Left="90"
       Height="120" Width="120" StrokeThickness="2"
       Stroke="Blue"/>
   </Canvas>
</Page>
```

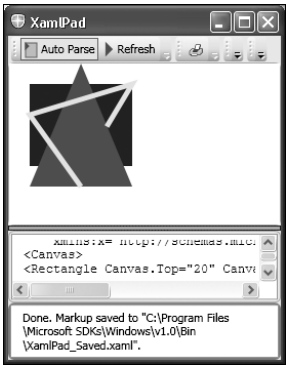


◄ **Figure 11-3 :** Un cercle

Vous pouvez également utiliser les formes `Rectangle`, `Polygon` et `Polyline`.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <Canvas>
      <Rectangle Canvas.Top="20" Canvas.Left= "20"
       Width="100" Height="80" Fill="Blue"/>
      <Polygon Fill="Green">
         <Polygon.Points>
```

```
            <Point X="70" Y="0"/>
            <Point X="20" Y="120"/>
            <Point X="120" Y="120"/>
         </Polygon.Points>
      </Polygon>
      <Polyline Stroke="Yellow" StrokeThickness="5">
         <Polyline.Points>
            <Point X="70" Y="120"/>
            <Point X="20" Y="50"/>
            <Point X="120" Y="20"/>
            <Point X="108" Y="40"/>
            <Point X="120" Y="20"/>
         </Polyline.Points>
      </Polyline>
   </Canvas>
</Page>
```



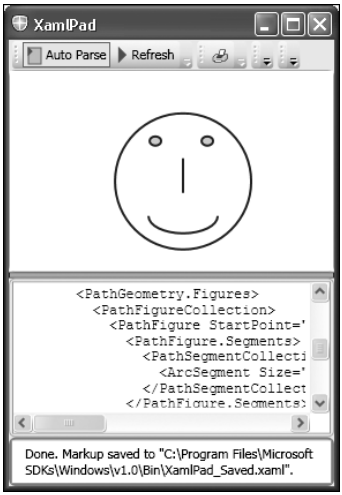
◄ **Figure 11-4 :** Les autres formes

Avec la classe `Polygon`, vous pouvez tracer n'importe quelle figure. La liste des points fournit les différents sommets du polygone. La classe `Polyline` est très similaire mais permet de réaliser des formes ouvertes. Les points fournis sont non plus les sommets mais simplement des points de passage reliés par des lignes droites.

En dehors des figures de base, qui sont le moyen le plus simple de dessiner, il est également possible de tracer des formes complexes en utilisant un chemin. Ce chemin sera une succession de segments de formes différentes. Dans un premier exemple, limitons le chemin à un segment en forme d'arc.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
```

```
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="120"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="165"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Canvas.Top="10" Canvas.Left="90"
       Height="120" Width="120" StrokeThickness="2"
       Stroke="Blue"/>
      <Path Stroke="Blue" StrokeThickness="2">
         <Path.Data>
            <PathGeometry>
               <PathGeometry.Figures>
                  <PathFigure StartPoint="120,100">
                     <ArcSegment Size="10,5" Point="180,100"/>
                  </PathFigure>
               </PathGeometry.Figures>
            </PathGeometry>
         </Path.Data>
      </Path>
   </Canvas>
</Page>
```



◀ **Figure 11-5 :** Un arc de cercle

Les segments sont rassemblés pour former une figure. L'attribut `SartPoint` de la balise `PathFigure` permet de définir le point de départ de la figure. Ensuite, les segments définissent le point d'arrivée suivant du trait. C'est pour cela que

l'on parle de chemin et non d'une collection de formes. Les autres attributs du segment permettent de définir la forme que le trait va prendre. Dans le cas d'un arc, c'est l'attribut `Size` qui va définir la courbure.

Bien sûr nous pouvons appliquer les techniques d'animation vues précédemment pour animer notre dessin.

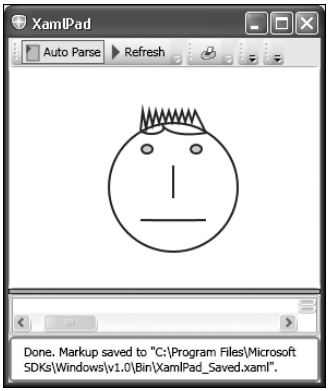
```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
   <Page.Triggers>
      <EventTrigger RoutedEvent="Page.Loaded">
         <EventTrigger.Actions>
            <BeginStoryboard>
               <Storyboard>
                  <ColorAnimation
                   Storyboard.TargetName="oeil1"
                   Storyboard.TargetProperty=
                   "(Ellipse.Fill).(SolidColorBrush.Color)"
                   From="LightBlue" To="Red"
                   Duration="00:00:7"
                   AutoReverse="True" />
                  <SizeAnimation
                   Storyboard.TargetName="bouche"
                   Storyboard.TargetProperty="Size"
                   From="100,5" To="10,5"
                   Duration="00:00:7"
                   AutoReverse="True" />
                </Storyboard>
             </BeginStoryboard>
          </EventTrigger.Actions>
      </EventTrigger>
   </Page.Triggers>
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="120"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Name="oeil1" Canvas.Top="30"
       Canvas.Left="165" Height="10" Width="12"
       StrokeThickness="2" Stroke="Blue"
       Fill="LightBlue"/>
      <Ellipse Canvas.Top="10" Canvas.Left="90"
       Height="120" Width="120" StrokeThickness="2"
       Stroke="Blue"/>
      <Path Stroke="Blue" StrokeThickness="2">
         <Path.Data>
```

```
            <PathGeometry>
               <PathGeometry.Figures>
                  <PathFigure StartPoint="120,100">
                     <ArcSegment x:Name="bouche"
                      Size="10,5" Point="180,100"/>
                  </PathFigure>
               </PathGeometry.Figures>
            </PathGeometry>
         </Path.Data>
      </Path>
   </Canvas>
</Page>
```

Il existe toute une panoplie de segments utilisables : ArcSegment, BezierSegment, LineSegment, PolyBezierSegment, PolyLineSegment, PolyQuadraticBezierSegment, QuadraticBezierSegment.

```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Page1" Height="122" Width="491">
   <Canvas>
      <Line X1="150" Y1="50" X2="150" Y2="80"
       StrokeThickness="2" Stroke="Blue"/>
      <Ellipse Canvas.Top="30" Canvas.Left="120"
       Height="10" Width="12" StrokeThickness="2"
       Stroke="Blue" Fill="LightBlue"/>
      <Ellipse Name="oeil1" Canvas.Top="30"
       Canvas.Left="165" Height="10" Width="12"
       StrokeThickness="2" Stroke="Blue"
       Fill="LightBlue"/>
      <Ellipse Canvas.Top="10" Canvas.Left="90"
       Height="120" Width="120" StrokeThickness="2"
       Stroke="Blue"/>
      <Path Stroke="Blue" StrokeThickness="2">
         <Path.Data>
            <PathGeometry>
               <PathGeometry.Figures>
                  <PathFigure StartPoint="120,100">
                     <ArcSegment x:Name="bouche"
                      Size="10,5" Point="180,100"/>
                  </PathFigure>
                 <PathFigure StartPoint="120,18">
                     <ArcSegment Size="10,5"
                      Point="140,13"/>
                     <BezierSegment Point1="140,15"
                      Point2="150,25" Point3="180,20"/>
                     <LineSegment Point="170,0"/>
```

```
                    <PolyLineSegment
                     Points="167,10,164,0,161,10,158,0,155,10,152,0,149,10,
                     ➥ 146,0,143,10,140,0,137,11,134,0,131,13,128,0,125,16,
                     ➥ 122,0,120,18">
                     </PolyLineSegment>
                  </PathFigure>
               </PathGeometry.Figures>
            </PathGeometry>
         </Path.Data>
      </Path>
   </Canvas>
</Page>
```



◄ **Figure 11-6 :** Divers segments

`PathGeometry` n'est pas la seule balise que vous puissiez placer dans le nœud `Path.Data`. Vous pouvez par exemple utiliser `LineGeometry`, `RectangleGeometry` ou encore `EllipseGeometry`. Ces classes sont très semblables aux formes correspondantes dont nous avons déjà parlé excepté qu'elles ont besoin d'une autre classe comme `Path` pour pouvoir être affichées.

11.2 Le dessin en 3D

XAML ne se contente pas de dessiner en deux dimensions mais est également capable de dessiner en trois dimensions. Attention, pour pouvoir utiliser ces fonctionnalités, il est nécessaire de posséder des notions de dessin en trois dimensions. XAML vous fournit en effet les instructions adéquates mais il est nécessaire de connaître les techniques spécifiques à ce domaine d'activité. C'est pourquoi nous nous limiterons à quelques notions de base et à un exemple simple.

Tout d'abord, pour réaliser des éléments en 3D, vous devez définir un conteneur spécifique, il s'agit de `Viewport3D`. Il sera alors nécessaire de définir deux parties distinctes. D'une part la caméra, qui détermine la projection 3D sur la

surface 2D, c'est-à-dire l'angle de vue sur le modèle, d'autre part le modèle lui-même. Dans l'exemple, le modèle est composé de deux sous-modèles : tout d'abord un éclairage. Celui-ci va permettre de définir un spot lumineux qui va avoir pour effet de modifier l'éclairage (les couleurs) du contenu selon l'angle avec lequel il est présenté. Par exemple, dans le cas d'un cube, la face dirigée vers le spot sera plus lumineuse. Ensuite, nous définissons le contenu réel de la scène : ici un cube. Bien sûr, vous pouvez multiplier les objets présentés et même placer plusieurs spots.

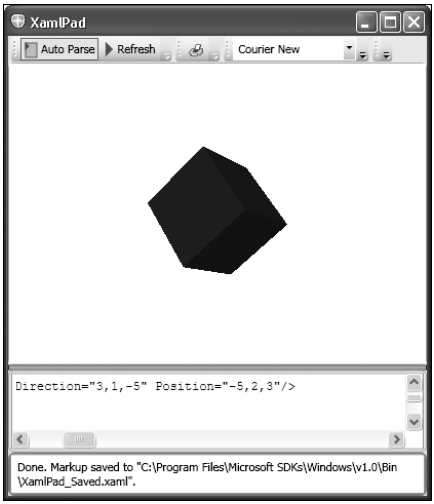
Pour définir la caméra, nous allons utiliser la balise `PrespectiveCamera` et définir l'angle de vue grâce aux propriétés `LookDirection` et `UpDirection`. L'attribut `Position` définit la position de la caméra dans la scène. Les positions et les directions sont données sous forme d'un point 3D (classe `Point3D`), c'est-à-dire sous forme d'un système de coordonnées sur trois axes, X et Y étant les axes habituels alors que Z donne la profondeur. Si Z vaut 0, nous retrouvons le plan 2D.

Pour définir le spot, nous allons utiliser la balise `DirectionalLight`. Ici, le spot est défini dans le même sens que la projection de manière à faire ressortir la face avant. La couleur de la lumière est blanche pour conserver toute sa clarté à la scène.

Le cube lui-même est construit en deux parties ; d'une part le cube, défini en utilisant la balise `MeshGeometry3D` et en déterminant les points de passage, d'autre part la surface à afficher sur la forme ainsi définie. Nous afficherons ici une simple surface bleue. Vous pourriez tout aussi facilement afficher une image.

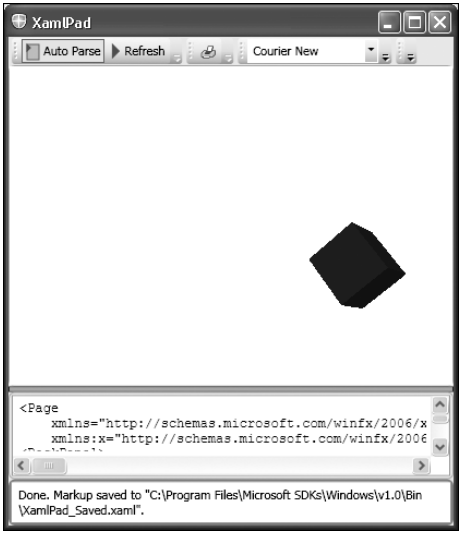
```
<Page
 xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <DockPanel>
      <Viewport3D>
         <Viewport3D.Camera>
            <PerspectiveCamera LookDirection="5,-2,-3"
             UpDirection="3,1,-5" Position="-5,2,3"/>
         </Viewport3D.Camera>
         <ModelVisual3D>
            <ModelVisual3D.Children>
               <ModelVisual3D>
                  <ModelVisual3D.Content>
                     <DirectionalLight Color="White"
                      Direction="5,-3,-2" />
                  </ModelVisual3D.Content>
               </ModelVisual3D>
               <ModelVisual3D>
                  <ModelVisual3D.Content>
                     <GeometryModel3D>
```

```
                        <GeometryModel3D.Geometry>
                            <MeshGeometry3D Positions="-0.5
,-0.5,0.5 0.5,-0.5,0.5 -0.5,0.5,0.5 0.5,-0.5,0.5 0.5,0.5
,0.5 -0.5,0.5,0.5 -0.5,0.5,0.5 0.5,0.5,0.5 -0.5,0.5
,-0.5 0.5,0.5,0.5 0.5,0.5,-0.5 -0.5,0.5,-0.5 -0.5,0.5
,-0.5 0.5,0.5,-0.5 -0.5,-0.5,-0.5 0.5,0.5,-0.5 0.5,-0.5
,-0.5 -0.5,-0.5,-0.5 -0.5,-0.5,-0.5 0.5,-0.5,-0.5 -0.5
,-0.5,0.5 0.5,-0.5,-0.5 0.5,-0.5,0.5 -0.5,-0.5,0.5 0.5
,-0.5,0.5 0.5,-0.5,-0.5 0.5,0.5,0.5 0.5,-0.5,-0.5 0.5,0.5
,-0.5 0.5,0.5,0.5 -0.5,-0.5,-0.5 -0.5,-0.5,0.5 -0.5,0.5
,-0.5 -0.5,-0.5,0.5 -0.5,0.5,0.5 -0.5,0.5,-0.5"/>
                        </GeometryModel3D.Geometry>
                        <GeometryModel3D.Material>
                            <DiffuseMaterial>
                                <DiffuseMaterial.Brush>
                                    <SolidColorBrush
                                     Color="Blue"
                                     Opacity="1.0" />
                                </DiffuseMaterial.Brush>
                            </DiffuseMaterial>
                        </GeometryModel3D.Material>
                    </GeometryModel3D>
                </ModelVisual3D.Content>
            </ModelVisual3D>
        </ModelVisual3D.Children>
    </ModelVisual3D>
</Viewport3D>
</DockPanel>
</Page>
```



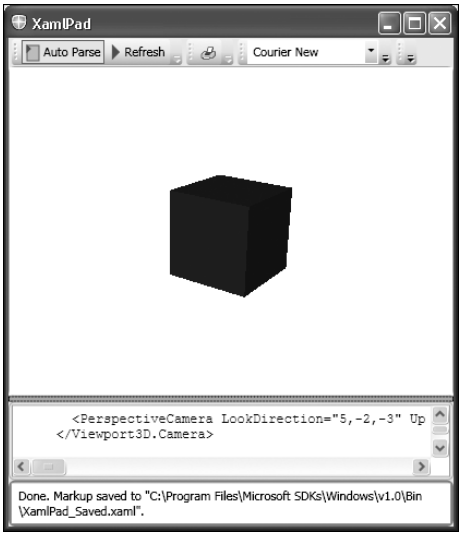
◄ **Figure 11-7 :** Un cube

Si vous modifiez par exemple la position de la caméra en remplaçant -5 par -8, vous l'éloignez ainsi de la scène. Le résultat est que le cube se déplace vers le bas et vers la droite mais aussi que sa taille se réduit.



◄ **Figure 11-8 :** Un cube

Par contre en transformant le 3 en 8 dans la balise `UpDirection`, vous allez modifier l'angle de vue.



◄ **Figure 11-9 :** Un cube

À vous de déplacer la caméra pour obtenir ce que vous désirez. Bien sûr, vous pouvez réaliser des animations en modifiant ces différents paramètres.

11.3 Checklist

Les points essentiels qui ont été vus dans ce chapitre sont :

- le dessin en 2D avec les formes comme `Ellipse`, `Rectangle`, `Polygone`... ;
- le dessin en 2D en utilisant des suites de segments comme `ArcSegment`, `LineSegment`, `BezierSegment`... ;
- l'animation de dessin en 2D ;
- les bases du dessin en 3D avec `Viewport3D`, `PerspectiveCamera`, `DirectionalLight`, et `MeshGeometry3D`.

Réaliser une application complète

Checklist 349

Pour parfaire notre maîtrise de XAML, il est maintenant temps de réaliser un exercice complet récapitulant un grand nombre de notions vues parfois en les abordant sous un aspect légèrement différent. Nous simulerons également une situation de travail collaboratif entre programmeur et designer.

Nous prendrons comme exemple une calculatrice. C'est une application dont le modèle métier est simple mais qui offre des possibilités de design nombreuses. C'est justement ce qu'il nous faut.

Le fait de choisir une application simple n'implique pas un certain respect des bonnes pratiques. Nous découperons donc notre application en deux couches. Nous ne développerons pas de couche de persistance des données, qui n'apporterait rien à l'étude du XAML.

La première partie de l'application sera réalisée par le programmeur, qui va réaliser sa couche métier. Notre couche se limite à une classe qui modélise la calculatrice. L'objectif n'étant pas l'étude de la modélisation, nous allons faire rapide et simple. Pour faire plaisir à ceux, nombreux, qui préfèrent C# mais aussi pour vous démontrer qu'il n'y a aucune différence dans XAML quand il est utilisé avec C# plutôt que VB.NET, nous réaliserons, vous l'aurez compris, le code .NET avec C#.

En premier, il faut créer une application WinFX Windows. Vous pouvez choisir un autre modèle mais vous devrez alors adapter la balise `root` du XAML et la définition de la classe dans le code associé.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Calculatrice
{
    class CalcBase
    {
```

Le membre `terme1` est destiné à recevoir le résultat affiché sur la calculatrice. Pour maintenir l'intégrité du système, cette valeur ne peut être modifiée que par la classe elle-même. C'est pourquoi la propriété qui y donne accès est en lecture seule.

```
        private decimal terme1;
        public decimal Terme1
        {
            get { return terme1; }
        }
```

Le membre terme2 est destiné à recevoir la valeur qui est entrée en vue de réaliser une opération. Nous aurions pu aussi la remplacer par des paramètres dans les méthodes.

```
private decimal terme2;
public decimal Terme2
{
    get { return terme2; }
    set { terme2 = value; }
}
```

Le membre memoire est destiné à recevoir la valeur stockée dans la mémoire de la calculatrice. L'accès à ce membre se fait uniquement *via* des méthodes. Il n'est donc pas exposé à l'extérieur.

```
private decimal memoire;
```

Le constructeur initialise les variables.

```
public CalcBase()
{
    memoire = 0;
    Reset();
}
```

La méthode Reset modélise l'utilisation de la touche *C*.

```
public void Reset()
{
    terme1 = 0;
    terme2 = 0;
}
```

```
public void Addition()
{
    terme1 += terme2;
    terme2 = 0;
}
```

```
public void Soustraction()
{
    terme1 -= terme2;
    terme2 = 0;
}
```

```
        public void Multiplication()
        {
            terme1 *= terme2;
            terme2 = 0;
        }
```

```
        public void Division()
        {
            if (terme2 != 0)
            {
                terme1 /= terme2;
                terme2 = 0;
            }
        }
```

La méthode `AjoutMemoire` modélise l'utilisation de la touche *M+*. Remarquez que c'est le résultat qui est mis en mémoire.

```
        public void AjoutMemoire()
        {
            memoire += terme1;
        }
```

La méthode `EffaceMemoire` modélise l'utilisation de la touche *MC*.

```
        public void EffaceMemoire()
        {
            memoire = 0;
        }
```

La méthode `RestitueMemoire` modélise l'utilisation de la touche *MR*. La valeur mémoire est restituée dans le deuxième terme de l'opération.

```
        public void RestitueMemoire()
        {
            terme2 = memoire;
        }
```

La méthode `EntreeNouvelleValeur` modélise l'instruction d'une première valeur dans un calcul.

```
        public void EntreeNouvelleValeur()
        {
            terme1 = terme2;
            terme2 = 0;
        }
    }
}
```

En bonne pratique, à ce stade, nous devrions écrire les tests unitaires que nous aurions dû prévoir à l'avance. Exceptionnellement, comme il n'apporte rien à l'apprentissage XAML, nous allons les oublier et passer directement à la couche de présentation.

```
<Window x:Class="Calculatrice.Window1" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Calculatrice"
 Height="365" Width="455"
 >
```

Nous allons utiliser une grille pour placer nos éléments.

```
   <Grid Height="330" Width="440">
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="0.25*" />
         <ColumnDefinition Width="0.25*" />
         <ColumnDefinition Width="0.25*" />
         <ColumnDefinition Width="0.25*" />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
         <RowDefinition Height="0.14*" />
      </Grid.RowDefinitions>
```

Pour rendre notre calculatrice un peu originale, nous allons utiliser une zone pour le résultat et une autre pour le terme. Notez que `terme2` est plus petit que résultat, cela permet de garder de la place pour d'autres informations.

```
      <Label Grid.Row="0"
       Name="resultat"
       Grid.ColumnSpan="4"
       Margin="5,5,5,5"
       Content="0"
       BorderBrush="Black"
       BorderThickness="1"
       HorizontalContentAlignment="Right"
       VerticalContentAlignment="Center"
       />
      <Label Grid.Row="1"
       Name="terme2"
       Grid.Column="1"
       Grid.ColumnSpan="3"
```

```
        Margin="5,5,5,5"
        Content="0"
        HorizontalContentAlignment="Right"
        VerticalContentAlignment="Center"
        BorderBrush="Black"
        BorderThickness="1"
        />
```

Nous aurons également besoin d'afficher un indicateur signalant quand une donnée est stockée dans la mémoire et allons aussi afficher l'opérateur qui a été sélectionné. Pour cela nous allons utiliser la cellule laissée libre par `terme2`.

```
      <WrapPanel Grid.Row="1">
         <Label Name="memoireActive"
          Grid.Column="0"
          Margin="5,5,5,5"
          Content=""
          VerticalContentAlignment="Center"
          VerticalAlignment="Center"
          Width="40"
          />
         <Label Grid.Row="1"
          Name="operateur"
          Grid.Column="0"
          Margin="5,5,5,5"
          Content=""
          VerticalContentAlignment="Center"
          VerticalAlignment="Center"
          />
      </WrapPanel>
```

Il nous reste maintenant à définir les boutons pour les différentes touches. Pour chacun d'entre eux, nous devrons associer une méthode à l'événement `Click`. Nous définirons ultérieurement les méthodes dans la classe.

```
      <Button Grid.Row="2"
       Grid.Column="0"
       Content="MC"
       Click="clickOnMC"/>
      <Button Grid.Row="2"
       Grid.Column="1"
       Content="MR"
       Click="clickOnMR"/>
      <Button Grid.Row="2"
       Grid.Column="2"
       Content="M+"
       Click="clickOnMemoirePlus"/>
      <Button Grid.Row="2"
       Grid.Column="3"
```

```
        Content="+"
        Click="clickOnPlus"/>
      <Button Grid.Row="3"
        Grid.Column="0"
        Content="1"
        Click="clickOn1"/>
      <Button Grid.Row="3"
        Grid.Column="1"
        Content="2"
        Click="clickOn2"/>
      <Button Grid.Row="3"
        Grid.Column="2"
        Content="3"
        Click="clickOn3"/>
      <Button Grid.Row="3"
        Grid.Column="3"
        Content="-"
        Click="clickOnMoins"/>
      <Button Grid.Row="4"
        Grid.Column="0"
        Content="4"
        Click="clickOn4"/>
      <Button Grid.Row="4"
        Grid.Column="1"
        Content="5"
        Click="clickOn5"/>
      <Button Grid.Row="4"
        Grid.Column="2"
        Content="6"
        Click="clickOn6"/>
      <Button Grid.Row="4"
        Grid.Column="3"
        Content="*"
        Click="clickOnMultiplication"/>
      <Button Grid.Row="5"
        Grid.Column="0"
        Content="7"
        Click="clickOn7"/>
      <Button Grid.Row="5"
        Grid.Column="1"
        Content="8"
        Click="clickOn8"/>
      <Button Grid.Row="5"
        Grid.Column="2"
        Content="9"
        Click="clickOn9"/>
      <Button Grid.Row="5"
        Grid.Column="3"
        Content="/"
        Click="clickOnDivision"/>
```

```
      <Button Grid.Row="6"
       Grid.Column="0"
       Content="C"
       Click="clickOnC"/>
      <Button Grid.Row="6"
       Grid.Column="1"
       Content="0"
       Click="clickOn0"/>
      <Button Grid.Row="6"
       Grid.Column="2"
       Content="."
       Click="clickOnPoint"/>
      <Button Grid.Row="6"
       Grid.Column="3"
       Content="="
       Click="clickOnEgal"/>
   </Grid>
</Window>
```

Maintenant que le code XAML est écrit, il nous reste à compléter le code .NET associé.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Calculatrice
{
    /// <summary>
    /// Interactions logiques pour Window1.xaml
    /// </summary>

    public partial class Window1 : Window
    {
```

Nous devons définir un membre de type `CalcBase` (notre objet métier).

```
        private CalcBase calc;
        public Window1()
        {
            calc = new CalcBase();
```

```
            InitializeComponent();
        }
```

La méthode suivante réalise un Reset de l'objet métier et rafraîchit l'affichage.

```
        private void clickOnC(object sender
                            , RoutedEventArgs e)
        {
            calc.Reset();
            operateur.Content = "";
            refreshValues();
        }
```

Les cinq méthodes suivantes provoquent le calcul et changent l'opérateur pour la prochaine opération.

```
        private void clickOnEgal(object sender
                               , RoutedEventArgs e)
        {
            operate();
            operateur.Content = "";
        }
        private void clickOnPlus(object sender
                               , RoutedEventArgs e)
        {
            operate();
            operateur.Content = "+";
        }
        private void clickOnDivision(object sender
                                   , RoutedEventArgs e)
        {
            operate();
            operateur.Content = "/";
        }
        private void clickOnMoins(object sender
                                , RoutedEventArgs e)
        {
            operate();
            operateur.Content = "-";
        }
        private void clickOnMultiplication(object sender
                                         , RoutedEventArgs e)
        {
            operate();
            operateur.Content = "*";
        }
```

Nous avons ensuite les méthodes liées aux boutons concernant la mémoire.

```
        private void clickOnMR(object sender
                               , RoutedEventArgs e)
        {
            calc.RestitueMemoire();
            refreshValues();
        }
        private void clickOnMC(object sender
                               , RoutedEventArgs e)
        {
            calc.EffaceMemoire();
            memoireActive.Content = "";
        }
        private void clickOnMemoirePlus(object sender
                                        , RoutedEventArgs e)
        {
            calc.AjoutMemoire() ;
            memoireActive.Content = "M";
        }
```

Les clics sur les boutons *0* à *9* ainsi que le point provoquent uniquement une modification du label `terme2`.

```
        private void clickOn1(object sender
                              , RoutedEventArgs e)
        {
            addDigit("1");
        }
        private void clickOn2(object sender
                              , RoutedEventArgs e)
        {
            addDigit("2");
        }
        private void clickOn3(object sender
                              , RoutedEventArgs e)
        {
            addDigit("3");
        }
        private void clickOn4(object sender
                              , RoutedEventArgs e)
        {
            addDigit("4");
        }
        private void clickOn5(object sender
                              , RoutedEventArgs e)
        {
            addDigit("5");
        }
        private void clickOn6(object sender
                              , RoutedEventArgs e)
```

```
        {
            addDigit("6");
        }
        private void clickOn7(object sender
                              , RoutedEventArgs e)
        {
            addDigit("7");
        }
        private void clickOn8(object sender
                              , RoutedEventArgs e)
        {
            addDigit("8");
        }
        private void clickOn9(object sender
                              , RoutedEventArgs e)
        {
            addDigit("9");
        }
        private void clickOn0(object sender
                              , RoutedEventArgs e)
        {
            addDigit("0");
        }
        private void clickOnPoint(object sender
                                  , RoutedEventArgs e)
        {
            addDigit(",");
        }
```

Attention

Problème de localisation

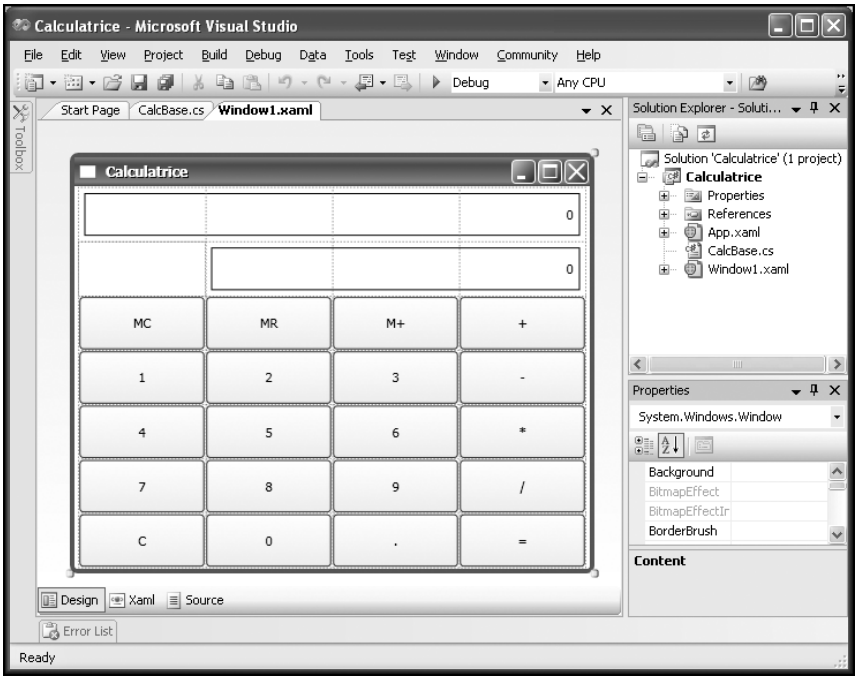
Le programme est prévu non pas pour tous les environnements mais uniquement pour une configuration avec la virgule décimale.

```
        private void addDigit(string car)
        {
            if (terme2.Content.ToString() == "0")
            {
                terme2.Content = car;
            }
            else
            {
                terme2.Content += car;
            }
        }
```

La méthode suivante détermine quelle opération doit être demandée à notre objet métier.

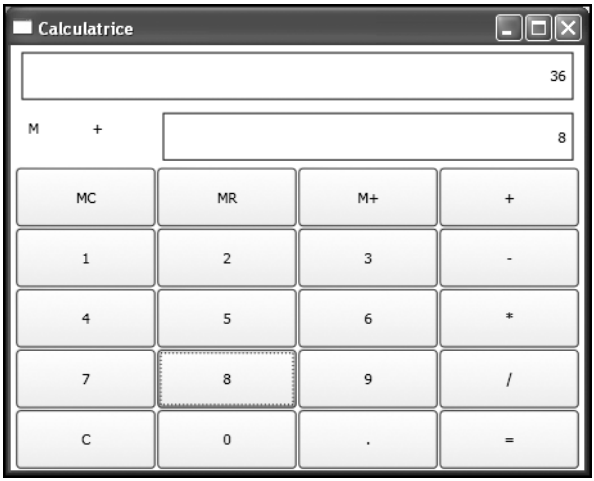
```
        private void operate()
        {
            calc.Terme2 = Convert.ToDecimal(terme2.Content);
            switch(operateur.Content.ToString())
            {
                case "+":
                {
                    calc.Addition();
                    break;
                }
                case "-":
                {
                    calc.Soustraction();
                    break;
                }
                case "/":
                {
                    calc.Division();
                    break;
                }
                case "*":
                {
                    calc.Multiplication();
                    break;
                }
                default:
                {
                    if (calc.Terme2 != 0)
                    {
                        calc.EntreeNouvelleValeur();
                    }
                    break;
                }
            }
            refreshValues();
        }
        private void refreshValues()
        {
            resultat.Content = calc.Terme1.ToString();
            terme2.Content = calc.Terme2.ToString();
        }
    }
}
```

Notre code est maintenant terminé. Si nous passons en mode *design*, nous recevrons l'écran suivant (ici l'écran présenté sous Visual Studio 2005 Team Edition).



▲ **Figure 12-1 :** Le code vu en mode design

Nous pouvons maintenant tester l'application, qui est totalement fonctionnelle.



▲ **Figure 12-2 :** La calculatrice dans son design d'origine

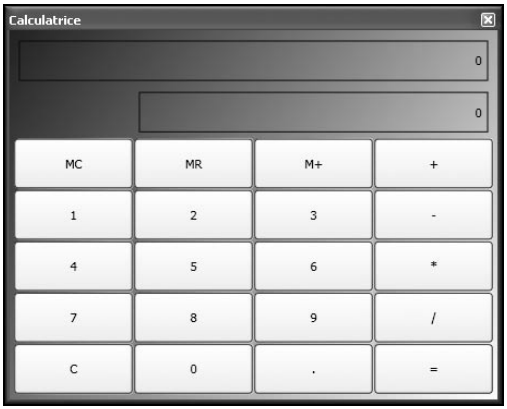
Le programmeur a développé la calculatrice en lui donnant une interface sombre mais parfaitement fonctionnelle. Il peut maintenant passer la main au designer, qui va améliorer la présentation mais sans devoir intervenir dans les mécanismes. Pour nous, il s'agira d'un changement de rôle.

Dans cette phase, nous n'interviendrons plus que sur le code XAML.

La première étape consistera à changer le fond d'écran ainsi qu'à donner au cadre Windows un aspect boîte d'outils. Nous allons utiliser les propriétés `WindowStyle` et `Background` de la class `Window`.

```
<Window x:Class="Calculatrice.Window1" xmlns=
 "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Calculatrice"
 Height="365" Width="455"
 WindowStyle="ToolWindow"
 >
   <Window.Background >
      <LinearGradientBrush StartPoint="0,0"
                           EndPoint="1,1">
         <LinearGradientBrush.GradientStops>
            <GradientStop Color="Blue" Offset="0" />
            <GradientStop Color="White" Offset="1" />
         </LinearGradientBrush.GradientStops>
      </LinearGradientBrush>
   </Window.Background>
```

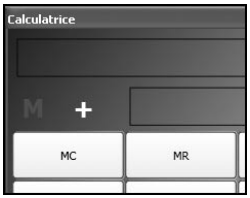
Comme fond, nous avons réalisé un dégradé bleu dans la diagonale de la fenêtre.



▲ **Figure 12-3 :** Dégradé en fond de fenêtre

Ensuite, nous allons agrandir la taille de l'opérateur et de l'indicateur pour la mémoire.

```
<Label
 Grid.Column="0"
 Name="memoireActive"
 Margin="5,5,5,5"
 Content=""
 VerticalContentAlignment="Center"
 VerticalAlignment="Center"
 Width="40"
 TextBlock.FontSize="24"
 TextBlock.FontWeight="UltraBlack"
 TextBlock.Foreground="Red"
 />
<Label Grid.Row="1"
 Name="operateur"
 Grid.Column="0"
 Margin="5,5,5,5"
 Content=""
 VerticalContentAlignment="Center"
 VerticalAlignment="Center"
 TextBlock.FontSize="24"
 TextBlock.FontWeight="UltraBlack"
 TextBlock.Foreground="White"
 />
```



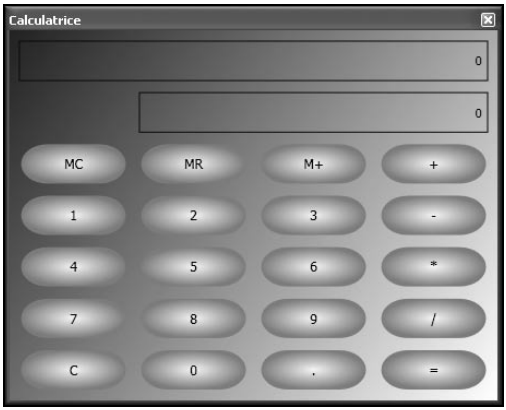
◄ **Figure 12-4 :** L'opérateur et l'indicateur mémoire

Passons maintenant à quelque chose de plus consistant, la présentation des boutons. Pour modifier cette dernière, nous allons utiliser un style. Le style sera automatiquement appliqué à tous les boutons. La présentation du bouton est complètement transformée et remplacée par un contrôle `Border` dont le fond est réalisé avec un dégradé radial. La balise `ContentPresenter` permet de gérer le texte contenu.

Le code suivant doit être inséré après la balise de fin `Window.Background`.

```
<Window.Resources>
   <Style TargetType="{x:Type Button}">
      <Setter Property="Button.Template">
         <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
               <Border BorderThickness="2"
                CornerRadius="25"
```

```
                Width="100" Height="40">
                  <Border.Background>
                     <RadialGradientBrush >
                       <RadialGradientBrush.GradientStops>
                           <GradientStop
                            Color="LavenderBlush"
                            Offset="0" />
                           <GradientStop Color="SlateGray"
                            Offset="1" />
                       </RadialGradientBrush.GradientStops>
                     </RadialGradientBrush>
                  </Border.Background>
                  <ContentPresenter
                   VerticalAlignment="Center"
                   HorizontalAlignment="Center"
                   TextBlock.FontWeight="Medium"
                   TextBlock.FontSize="12" />
                </Border>
              </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
<Window.Resources>
```



▲ **Figure 12-5 :** Les nouveaux boutons

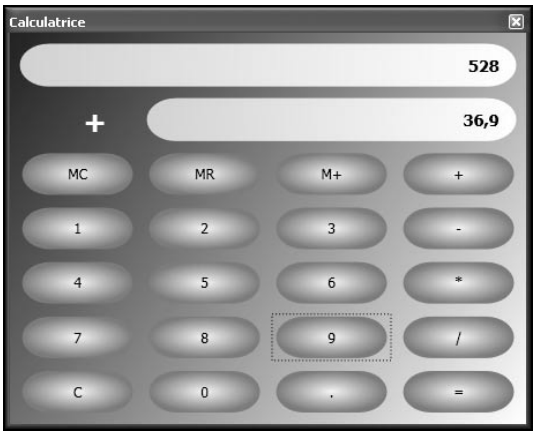
Après les boutons, il est temps de s'attaquer au changement des `Label`. Nous allons à nouveau utiliser un style mais nous ne pourrons cette fois l'appliquer automatiquement car seuls deux `Label` doivent être modifiés. Nous utiliserons donc un style nommé.

Le code suivant doit être inséré à la fin du nœud contenant les ressources de `Window`.

```
<Style x:Key="Affichage">
   <Setter Property="Label.Template">
      <Setter.Value>
         <ControlTemplate TargetType="{x:Type Label}">
            <Border CornerRadius="25">
               <Border.Background>
                  <LinearGradientBrush>
                     <LinearGradientBrush.GradientStops>
                        <GradientStop x:Name="debut"
                         Color="LightGray" Offset="0" />
                        <GradientStop x:Name="fin"
                         Color="White" Offset="1" />
                     </LinearGradientBrush.GradientStops>
                  </LinearGradientBrush>
               </Border.Background>
               <ContentPresenter VerticalAlignment="Center"
                HorizontalAlignment="Right"
                Margin="0,0,15,0" TextBlock.FontSize="14"
                TextBlock.FontWeight="UltraBlack"/>
            </Border>
         </ControlTemplate>
      </Setter.Value>
   </Setter>
</Style>
```

Nous devons maintenant spécifier au niveau des `Label` `resultat` et `terme2` qu'ils doivent utiliser ce nouveau style. Pour cela, il suffit d'ajouter dans la balise l'attribut suivant :

```
Style="{StaticResource Affichage}"
```



▲ **Figure 12-6 :** Le nouvel affichage des valeurs

La présentation commence désormais à prendre forme. Nous allons maintenant réaliser quelques petites finitions gadgets pour totalement personnaliser notre interface. Comme vous allez le voir, les détails sont souvent aussi les éléments les plus coûteux en temps de mise en œuvre.

L'idée de ce qui va suivre est de mettre en avant le bouton sur lequel l'utilisateur est positionné. La taille du texte affiché va être agrandie pour le bouton sur lequel la souris est positionnée. Au lieu de se contenter d'un simple passage d'un état à l'autre, nous allons réaliser un effet de fondu.

Pour réaliser cela, nous aurons évidemment besoin d'utiliser les triggers et les animations.

```
<Window.Resources>
   <Style TargetType="{x:Type Button}">
      <Setter Property="Button.Template">
…
            <ContentPresenter VerticalAlignment="Center"
             HorizontalAlignment="Center"
             TextBlock.FontWeight="Medium"
             TextBlock.FontSize="12"
             x:Name="contentString" />
```

Nous devons nommer l'objet `ContentPresenter` car nous allons lui appliquer des transformations.

```
         </Border>
         <ControlTemplate.Triggers>
```

Le trigger sera déclenché lorsque la propriété `IsMouseOver` sera fixée à `true`.

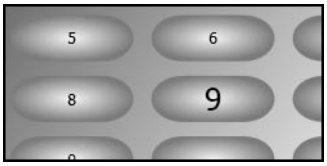
```
            <Trigger Property="IsMouseOver"
             Value="true">
```

Le trigger est composé de deux parties, une qui sera déclenchée lorsque la valeur passe à `true`, l'autre, lorsque la valeur devient autre, en l'occurrence `false`.

```
               <Trigger.EnterActions >
                 <BeginStoryboard>
                   <Storyboard>
                     <DoubleAnimation
                      Storyboard.TargetName="contentString"
                      Storyboard.TargetProperty=
                                  "(TextBlock.FontSize)"
                      From="12" To="24" Duration="0:0:0.5"/>
                   </Storyboard>
                 </BeginStoryboard>
               </Trigger.EnterActions>
```

Lors de la sortie, c'est l'animation inverse qui est réalisée.

```
            <Trigger.ExitActions >
              <BeginStoryboard>
                <Storyboard >
                  <DoubleAnimation
                   Storyboard.TargetName="contentString"
                   Storyboard.TargetProperty=
                                "(TextBlock.FontSize)"
                   From="24" To="12" Duration="0:0:0.5"/>
                </Storyboard>
              </BeginStoryboard>
            </Trigger.ExitActions>
          </Trigger>
        </ControlTemplate.Triggers>
      </ControlTemplate>
```



◀ **Figure 12-7 :** La mise en évidence du bouton

Pour aller plus loin dans cette mise en évidence, nous allons également faire passer le poids du caractère de `Medium` à `UltraBlack` et inversement.

Pour cela, nous devons modifier le trigger précédemment défini en y ajoutant une nouvelle animation à l'entrée et à la sortie.

Le trigger d'entrée devient maintenant :

```
<Trigger.EnterActions>
   <BeginStoryboard>
      <Storyboard>
         <DoubleAnimation
          Storyboard.TargetName="contentString"
          Storyboard.TargetProperty="(TextBlock.FontSize)"
          From="12" To="24" Duration="0:0:0.5"/>
```

Il s'agit d'une propriété qui doit recevoir une valeur statique d'une structure. Nous devons donc utiliser les classes `ObjectAnimationUsingKeyFrames` et `DiscreteObjectKeyFrame`.

```
         <ObjectAnimationUsingKeyFrames
          Storyboard.TargetName="contentString"
          Storyboard.TargetProperty="(TextBlock.FontWeight)"
          >
            <DiscreteObjectKeyFrame
```

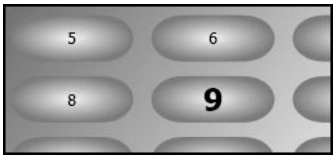
```
            KeyTime="0:0:0"
            Value="{x:Static FontWeights.Medium}" />
```

Pour fixer la valeur, il faut utiliser la syntaxe particulière avec `x:Static`.

```
          <DiscreteObjectKeyFrame
           KeyTime="0:0:0.5"
           Value="{x:Static FontWeights.UltraBlack}" />
        </ObjectAnimationUsingKeyFrames>
     </Storyboard>
  </BeginStoryboard>
</Trigger.EnterActions>
```

Quant à lui, le trigger de sortie devient :

```
<Trigger.ExitActions>
   <BeginStoryboard>
      <Storyboard>
         <DoubleAnimation
          Storyboard.TargetName="contentString"
          Storyboard.TargetProperty="(TextBlock.FontSize)"
          From="24" To="12" Duration="0:0:0.5"/>
         <ObjectAnimationUsingKeyFrames
          Storyboard.TargetName="contentString"
          Storyboard.TargetProperty="(TextBlock.FontWeight)"
          >
            <DiscreteObjectKeyFrame
             KeyTime="0:0:0"
             Value="{x:Static FontWeights.UltraBlack}" />
            <DiscreteObjectKeyFrame
             KeyTime="0:0:0.5"
             Value="{x:Static FontWeights.Medium}" />
         </ObjectAnimationUsingKeyFrames>
      </Storyboard>
   </BeginStoryboard>
</Trigger.ExitActions>
```



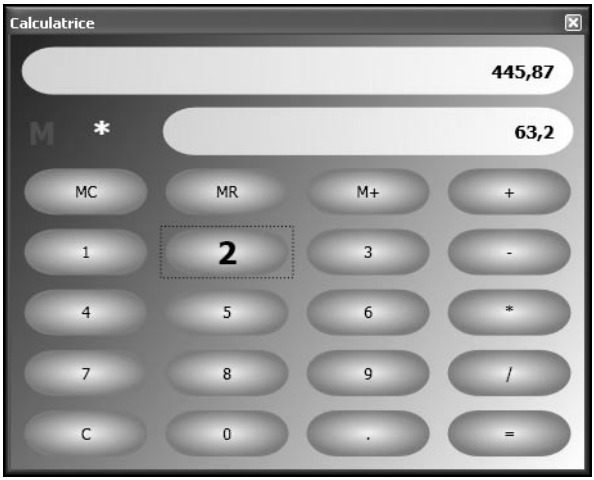
◄ **Figure 12-8 :** Encore plus de mise en évidence du bouton

Pour terminer, nous allons ajouter un effet sonore. Un clic sera émis lorsque les boutons de la calculatrice seront utilisés. Plutôt qu'ajouter cela dans le style, nous allons ajouter un trigger au niveau de la grille. Il s'agira d'un `Event Trigger` placé grâce à la propriété `RoutedEvent` sur l'événement `Button.Click`.

```
<Grid Height="330" Width="440">
   <Grid.Triggers>
     <EventTrigger RoutedEvent="Button.Click">
       <EventTrigger.Actions>
```

La balise SoundPlayerAction joue le fichier sonore spécifié dans la source.

```
         <SoundPlayerAction
          Source="c:\windows\media\Windows XP Infobulle.wav"
         />
       </EventTrigger.Actions>
     </EventTrigger>
   </Grid.Triggers>
…
```



▲ **Figure 12-9 :** L'exemple complet

Nous sommes maintenant au bout de cet exercice. Comme vous avez pu le constater, le fait d'avoir utilisé C# n'a rien changé au niveau XAML. Pour sa part, le designer n'a à aucun moment eu besoin d'accéder au code .NET.

12.1 Checklist

Dans le cadre de cet exercice, les notions principales vues ou revues sont :

- l'utilisation d'une grille pour positionner les contrôles ;
- la gestion des événements ;
- l'utilisation des styles ;

- la réalisation des dégradés avec les classes de type `GradientBrush` ;
- la modification de la présentation d'un contrôle en utilisant `ControlTemplate` et `ControlPresenter` ;
- l'utilisation des triggers et particulièrement en utilisant `EnterActions` et `ExitActions` ;
- l'utilisation des triggers de type `EventTrigger` ;
- l'utilisation des animations et tout particulièrement des animations sur des propriétés de type `object` et des valeurs de type statique.

Annexes

XAML sur le Web 352
Glossaire 359
Schéma d'héritage des différentes classes Visual 363
Résumé des classes et des attributs utilisés ... 368
Classes autorisées dans la zone internet 409
Liste des touches de raccourcis pour les commandes d'édition 411
Liste des classes par catégories 413
Liste des couleurs prédéfinies 415

13.1 XAML sur le Web

Si vous recherchez des informations sur le Web, voici quelques bonnes adresses. Malheureusement, nombreuses parmi elles sont en anglais.

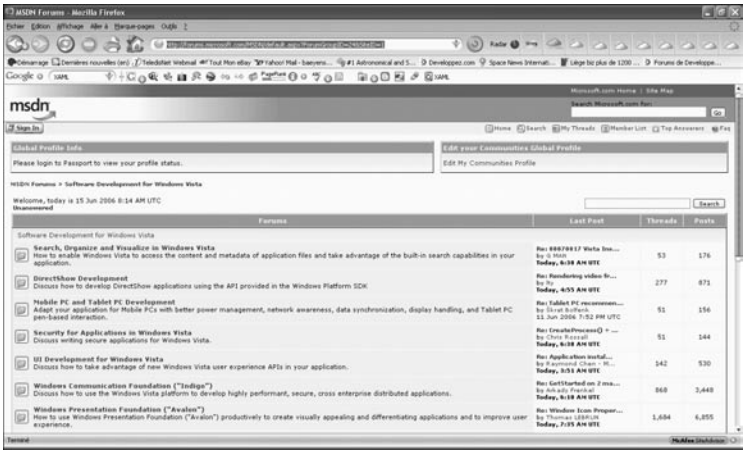
Tout d'abord, l'incontournable, le site officiel de Microsoft dédié à la technologie Framework 3.0, qui inclut WPF.

▲ **Figure 13-1 :** http://msdn.microsoft.com/winfx/

Toujours sur MSDN, vous pourrez retrouver l'incontournable aide en ligne. Le lien donné pointe sur Microsoft France mais, à l'heure actuelle, la documentation est toujours en anglais.

▲ **Figure 13-2 :** http://windowssdk.msdn.microsoft.com/fr-fr/library/

Si vous rencontrez des problèmes avec WPF et que vous vous débrouilliez en anglais, vous pouvez utiliser le Forum officiel.



▲ **Figure 13-3 :** http://forums.microsoft.com/MSDN/default.aspx?ForumGroupID=24&SiteID=1

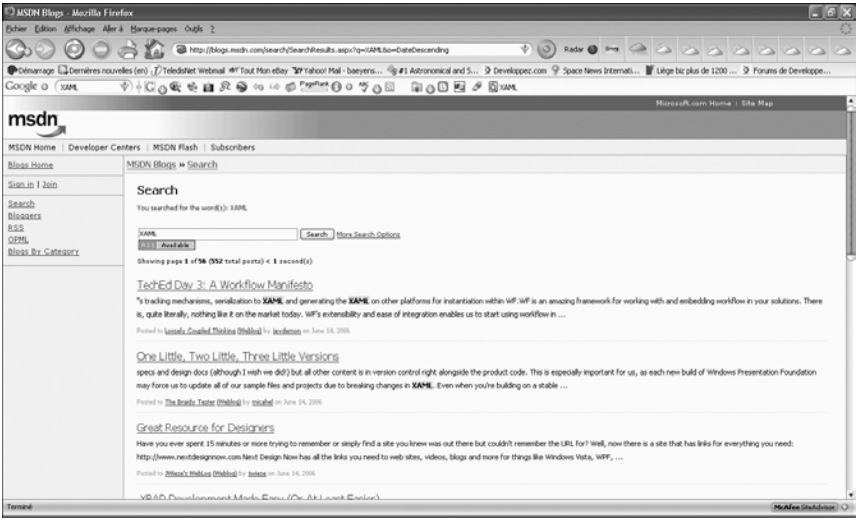
N'oubliez pas l'autre site sur le Framework 3.0. Il recèle un grand nombre d'exemples complets sur les différentes techniques de WinFX. Malgré son adresse, il s'agit bel et bien d'un site *made in Microsoft*.

▲ **Figure 13-4 :** http://wpf.netfx3.com/

Vous pouvez aussi retrouver WPF en vidéo avec Channel 9. Sur ce site, vous retrouverez des interviews et des conférences sur le sujet. C'est en général dans ces interviews ou dans les blogs des membres de l'équipe de développement que vous trouverez les informations les plus récentes.

▲ **Figure 13-5 :** http://channel9.msdn.com/Media/?TagID=2

En ce qui concerne les blogs, il n'y a pas de lien particulier, à vous d'utiliser la fonction de recherche. Vous pouvez par exemple essayer de chercher sur XAML ou encore WPF.



▲ **Figure 13-6 :** http://blogs.msdn.com/default.aspx

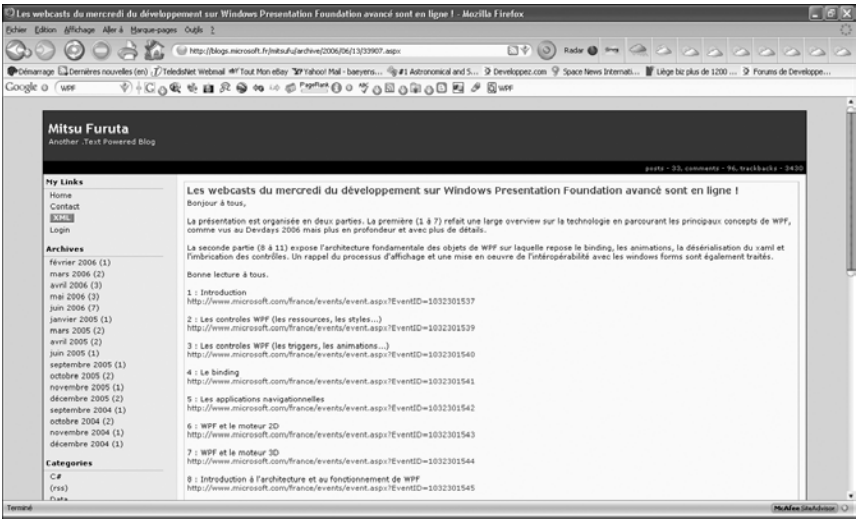
Heureusement, Microsoft France met également à notre disposition un nombre important d'informations depuis son site. Le principal site de Microsoft France parlant du sujet est le site dédié à Windows Vista.

▲ **Figure 13-7 :** http://www.microsoft.com/france/msdn/windowsvista/default.mspx

Ne ratez pas également les Webcast en français qui vous y sont proposés. Attention, pour visualiser ces Webcast, une inscription est obligatoire !

▲ **Figure 13-8 :** http://www.microsoft.com/france/msdn/webcasts/webcasts-DevWindows.mspx

Dans les deux sites cités ci-dessus, vous verrez souvent apparaître le nom de Mitsu Furuta, relation technique avec les développeurs. Son blog est incontournable pour rester informé de l'actualité.



▲ **Figure 13-9 :** http://blogs.microsoft.fr/mitsufu/

Microsoft n'est pas le seul à fournir de l'information sur le sujet. Les sites communautaires sont aussi une bonne source d'informations. Les articles que vous y trouverez sont souvent plus adaptés pour un débutant ou plus ciblés à une problématique particulière. Ils offrent en plus généralement un forum où vous pourrez poser vos questions.

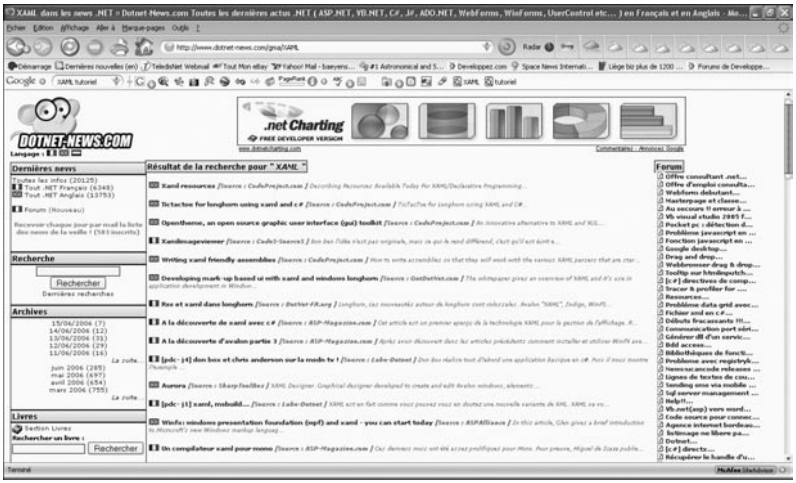
Le premier d'entre eux que je voudrais citer est Developpez.com. Bien qu'il n'y ait pas de rubrique spécifique à WinFX, vous y retrouverez des articles intéressants écrits par les membres de la rédaction et par exemple un très bel article de Thomas Lebrun (**http://morpheus.developpez.com/windows-presentation-foundation/**). Le forum est très fréquenté et, si vous avez des questions, vous y obtiendrez très certainement la réponse. De nombreux blogs vous tiendront informé de l'actualité informatique en général et donc de celle liée à XAML aussi. Vous y trouverez également un espace TV où vous pourrez entre autres visionner les Devdays 2006, dans lesquels il a été abondamment question de WPF.

▲ **Figure 13-10 :** http://dotnet.developpez.tv/devdays2006/

Le site Asp-Php.net a quant à lui créé une rubrique à part entière pour la technologie XAML. Cette rubrique est encore peu fournie, gageons qu'elle va s'étoffer au fil du temps et deviendra rapidement une très bonne source d'information.

▲ **Figure 13-11 :** http://www.asp-php.net/tutorial/xaml/index.php

Si vous voulez avoir un aperçu rapide de ce qui existe sur le Web concernant XAML, vous pouvez vous rendre sur le site Dotnet-news.com et faire une recherche sur XAML. Bien sûr, la liste ne sera pas exhaustive mais, toutefois, les principaux sites communautaires y sont référencés.

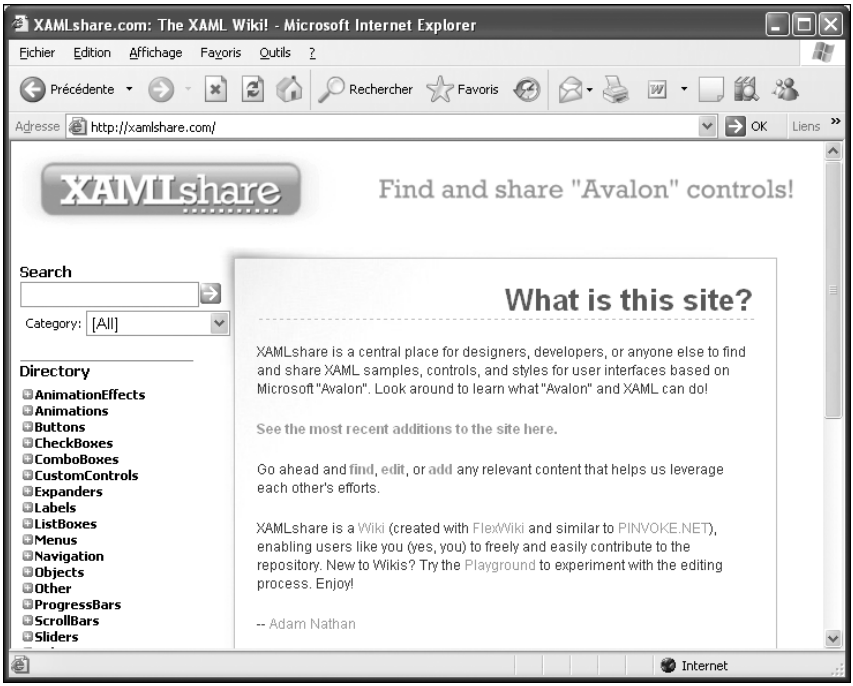


▲ **Figure 13-12 :** http://www.dotnet-news.com/gma/XAML

En ce qui concerne les sites communautaires anglophones, nous retrouvons le très classique Code Project, où vous pourrez d'ores et déjà trouver un grand nombre de ressources.

▲ **Figure 13-13 :** http://www.codeproject.com/

Mais n'hésitez pas à vous rendre sur le site de XAMLShare, beaucoup plus prometteur encore en terme de partage de ressources. Il regorge déjà de code très intéressant dont vous pourrez vous inspirer pour résoudre les problèmes que vous rencontrerez.



▲ **Figure 13-14 :** http://xamlshare.com/

Cette liste n'est évidemment pas exhaustive et est de plus susceptible d'évoluer fortement avec le temps. Toutefois, ces adresses devraient vous permettre d'une part de suivre l'actualité liée à XAML, qui va très probablement rester très abondante jusqu'à la sortie de Visual Studio 2007 et du Framework 3.0, et d'autre part de trouver des articles et des exemples ainsi qu'une assistance de la communauté qui ira croissante avec le temps et l'expérience des uns et des autres. Quant à moi, mon site se trouve à l'adresse **jab.developpez.com**.

13.2 Glossaire

Dans ce glossaire, vous trouverez la définition de termes rencontrés dans ce livre mais également de termes que vous rencontrerez dans l'aide de WinFX et qui semblaient pertinents d'expliquer.

API : Abréviation anglaise d'*Application Program Interface*. Il s'agit d'un ensemble de fonctions et/ou de classes permettant d'interagir avec une autre application. Windows pouvant être considéré comme une application, il possède son propre API.

ASP.NET : Technologie de Microsoft faisant partie du Framework .NET et destinée à construire des applications web. L'application est exécutée sur le serveur et génère du contenu HTML qui est envoyé vers le client. Le client doit uniquement disposer d'un navigateur. La logique du programme peut être écrite dans n'importe quel langage .NET comme C# ou VB.NET.

Assembly : Un assembly est un ensemble de fichiers déployé comme une unité et compilé en un bloc. Généralement, les notions d'assembly et de dll se confondent.

Attribut : Il s'agit d'une propriété d'une classe ou, dans le cadre XML, d'une propriété du nœud XML.

Balise : Une balise est un élément permettant de structurer un fichier XML dans notre cas. Il existe deux types de balises : une balise ouvrante et une balise fermante.

BAML : Abréviation anglaise de *Binary Application Markup Language*. Objet binaire obtenu après compilation d'un fichier XAML.

Classe : C'est un ensemble de propriétés et de méthodes regroupées dans une même entité et qui sont en relation avec un même concept. Une classe est une entité abstraite qui sert de définition pour les objets. On peut faire le rapprochement entre les concepts de classe et d'objet et les concepts de type de données et de donnée.

CLR : Abréviation anglaise de *Common Language Runtime*. C'est le moteur d'exécution des applications .NET. En effet, après compilation, le code .NET est transformé en code intermédiaire MSIL et non en code natif.

Code-Behind : Code .NET (VB.NET, C#...) contenant la classe qui implémente la logique pour un fichier XAML.

Code managé : Code exécuté par la CLR et non directement par le système d'exploitation.

Code non managé : Code exécuté directement par le système d'exploitation. Il est aussi appelé code natif.

Collection : Ensemble d'objets, de données généralement du même type.

Contrôle : C'est un composant du framework représenté par une classe et qui offre des capacités en terme d'interface utilisateur. Par exemple une `TextBox`.

Courbe de Bezier : Courbe calculée mathématiquement. Elle est définie par un ensemble de points de contrôle. Par exemple, la courbe cubique de Bezier est définie par les extrémités et deux points de passage.

Data binding : Le data binding, *liaison aux données* en français, est le terme technique généralement utilisé lorsqu'un mécanisme est mis en place pour réaliser une liaison automatique entre la source de données (qu'elle soit un fichier ou un objet métier) et le contrôle correspondant dans l'interface utilisateur. Ce mécanisme est normalement pris en charge par le langage utilisé, en l'occurrence ici le Framework .NET.

Événement : En programmation, un événement est un signal envoyé et qui peut être intercepté et traité par du code qui se met à l'écoute de cet événement. Le code ainsi associé sera exécuté quand ce signal est émis. Il est généralement associé à une action comme un clic de souris. Outre ceux déjà existants, vous pouvez créer vos propres événements.

Fixed Document : Format de document qui représente celui-ci exactement comme l'auteur l'a décidé.

Flow Document : Format de document qui représente celui-ci de façon à optimiser la lisibilité. L'affichage s'ajuste à l'environnement.

Framework : Ensemble de librairies de classes, de types de données et de tout autre élément propre à un environnement de développement.

Glyph : Série de segments utilisés pour représenter un mouvement.

Héritage : Mécanisme qui permet à une classe de disposer des propriétés et des méthodes de la classe dont elle hérite. Elle ne devra alors définir ou redéfinir les propriétés et méthodes qui lui sont propres.

IL : Voir MSIL.

Ink : Type de données représentant un trait.

Instance : Une instance est un objet d'une classe déterminée. Instance ou objet peuvent être considérés comme synonymes.

Instanciation : Action de créer une nouvelle instance d'une classe.

IntelliSense : Système permettant d'afficher dans un éditeur les éléments du langage correspondant à ce que vous avez déjà tapé, soit directement soit au travers d'une liste. Le but de l'IntelliSense est de faciliter le travail du développeur.

Interpolation linéaire : Dans le cadre de ce livre, il s'agit d'une méthode de transition entre deux états réalisée par un taux de changement constant pour chaque période de temps. La transition emprunte le chemin le plus court.

Interpolation splined : Il s'agit d'une méthode de transition entre deux états réalisée en suivant une courbe de Bezier.

Méthode : Une méthode est une fonction ou une procédure associée à une classe. Elle aura accès aux propriétés et membres de la classe sans devoir les recevoir en paramètre.

MSIL : Abréviation de *Microsoft Intermediate Language*. Code généré par le compilateur .NET. Quel que soit le langage que vous utilisez (VB.NET, C#...), le résultat sera du MSIL. Le MSIL sera à son tour compilé par le JIT *(just in time compiler*) pour être transformé en code natif et exécuté.

Nœud : Un nœud en XML est un ensemble compris entre une balise ouvrante et la balise fermante correspondante.

Objet : Un objet est la matérialisation d'une classe. Si *voiture* est une classe, votre *voiture* est un objet de la classe *voiture*.

Propriété : Une propriété est une variable spécifiquement associée à un objet.

Propriété attachée : Une propriété attachée est une propriété qui peut être attachée à n'importe quel objet dépendant de l'instance de la classe où elle est définie. Pour permettre cela, la classe doit contenir un accesseur statique (`Get` et `Set`) pour cette propriété attachée. Attention, il ne faut pas en déduire que la valeur est unique pour la classe ! Chaque objet qui utilise cette propriété conserve sa valeur particulière.

Resource : Ensemble d'informations non exécutable mais nécessaire à l'exécution du programme.

Template : Mot anglais utilisé dans le vocabulaire technique et dont la traduction habituelle est *modèle*. Le template peut représenter un modèle au sens le plus strict du mot, par exemple en ce qui concerne les .dot dans MS-Word. Dans l'environnement XAML comme dans d'autres, il décrit l'interface utilisateur pour l'objet auquel il est associé, ce qui permet de séparer contenu et présentation. Il ne faut pas confondre *template* et *style*. Le style permet de préciser des spécifications de l'interface comme la couleur ou la taille alors que le template permet de définir l'interface elle-même.

Transformation affine : Transformation linéaire suivie d'une translation.

Transformation linéaire : Transformation par rotation, changement d'échelle ou oblique.

Trigger : Mot technique anglais signifiant *déclencheur*. Initialement issu du monde des bases de données, il est également utilisé dans XAML. Un trigger est une petite procédure qui sera déclenchée automatiquement lorsque certaines conditions définies en même temps que le trigger sont rencontrées. Les mécanismes sont différents, mais le concept est assez semblable aux événements.

URI : *Uniform Resource Identifier*. Il s'agit de l'implémentation de la *RFC2396* de l'*Internet Engineering Task Force*.

WINFX : WinFX est un nouvel ensemble d'API destiné à remplacer les anciennes API Windows. Originalement créé pour Windows Vista, il sera également porté sur Windows XP.

WPF : Abréviation de *Windows Presentation Foundation*, précédemment connu sous le nom d'Avalon ; il s'agit d'un moteur d'affichage graphique pour Windows qui intègre entre autres nativement la 3D. WPF fait partie de WinFX. Il comprend également un langage déclaratif, XAML.

XML : Abréviation des termes anglais *eXtended Markup Langage* ou *Extensible Markup Language* selon les sources. Il s'agit d'un langage de description de données souvent utilisé pour la transmission d'information ou le stockage de faible volume. Comme le HTML, cette norme est héritée de SGML.

13.3 Schéma d'héritage des différentes classes Visual

Les schémas d'héritage ci-dessous ne sont pas les schémas complets. Ils reprennent principalement les classes qui ont été abordées dans l'ouvrage.

Schéma d'héritage des différentes classes Visual

<table>
<tr><th colspan="3">Schéma d'héritage des différentes classes Visual</th></tr>
<tr><th colspan="3">Classes d'héritage</th></tr>
<tr><td colspan="3">Visual</td></tr>
<tr><td colspan="3">UIElement</td></tr>
<tr><td colspan="3">FrameworkElement</td></tr>
<tr><td rowspan="6"></td><td colspan="2">Control</td></tr>
<tr><td></td><td>(voir tableau suivant)</td></tr>
<tr><td colspan="2">Decorator</td></tr>
<tr><td rowspan="2"></td><td>Border</td></tr>
<tr><td>ViewBox</td></tr>
<tr><td colspan="2">Panel</td></tr>
</table>

Schéma d'héritage des différentes classes Visual		
Classes d'héritage		
		Canvas
		DockPanel
		Grid
		StackPanel
		WrapPanel
	Image	
	MediaElement	
	Page	
		PageFunctionBase
	TextBlock	
	ViewPort3D	
	Shape	
		Ellipse
		Line
		Path
		Polyline
		Polygon
		Rectangle
	Popup	
	FixedPage	
	PageContent	
	ToolBarTray	

Le détail de l'héritage dans la branche Control.

Le détail de l'héritage dans la branche Control
Visual
UIElement
FrameworkElement

Le détail de l'héritage dans la branche Control

- Control
  - ContentControl
    - ButtonBase
      - Button
      - ToggleButton
        - CheckBox
        - RadioButton
      - RepeatButton
    - HeaderedContentControl
      - Expander
      - GoupBox
      - ToolBar
    - Frame
    - Label
    - ScrollViewer
    - ToolTip
    - Window
      - NavigationWindow
  - FlowDocumentScrollViewer
  - ItemsControl
    - Selector
      - ComboBox
      - ListBox
      - ListView
      - TabControl
    - TreeView
  - DocumentViewerBase
    - DocumentViewer
    - FlowdocumentPageViewer
  - Thumb
    - GridSplitter
  - ItemsControl
    - MenuBase
      - TabControl
  - TextBoxBase
    - RichTextBox
    - TextBox
  - RangeBase
    - Slider
  - PasswordBox

Schéma d'héritage des différentes classes ContentElement

Schéma d'héritage des différentes classes ContentElement				
ContentElement				
FrameworkContentElement				
	FixedDocument			
	FlowDocument			
	TextElement			
		Block		
			List	
			Paragraph	
			Section	
			Table	
		Inline		
			AnchoredBlock	
				Figure
				Floater
			LineBreak	
			Run	
			Span	
				Hyperlink
			ListItem	
			TableCell	
			TableRow	
			TablerowGroup	
		TableColumn		

Schéma d'héritage des différentes classes Freezable

Schéma d'héritage des différentes classes Freezable
Freezable
Animatable

Schéma d'héritage des différentes classes Freezable

- Brush
  - SolidColorBrush
  - GradiantBrush
    - LinearGradientBrush
    - RadialGradientBrush
  - TileBrush
    - ImageBrush
  - PathFigure
  - PathSegment
    - ArcSegment
    - BezierSegment
    - LineSegment
    - PolyBezierSegment
    - PolyLineSegment
    - QuadraticBezierSegment
    - PolyQuadraticBezierSegment
  - Pen
  - Timeline
    - AnimationTimeline
      - ColorAnimationBase
        - ColorAnimation
        - ColorAnimationUsingKeyFrame
      - DoubleAnimationBase
        - DoubleAnimation
        - DoubleAnimationUsingKeyFrame
    - TimelineGroup
      - ParallelTimeline
        - Storyboard
  - Material
    - DiffuseMaterial
  - Model3D
    - Light
      - DirectionalLight
  - Geometry3D
    - MeshGeometry3D
  - GradientStop
  - Camera
    - ProjectionCamera
      - PerstpectiveCamera
  - GeneralTransform
    - Transform
      - RotateTransform
      - ScaleTransform
      - SkewTransform
      - MatrixTransform
      - TranslateTransform
- DubleKeyFrame
  - SplineDoubleKeyFrame

13.4 Résumé des classes et des attributs utilisés

Classe ArcSegment

Classe ArcSegment	
Attribut	**Utilité**
`Size`	Définit les radiant X et Y déterminant les caractéristiques de la courbure
`Point`	Point d'arrivée
`SweepDirection`	Définit le sens de la courbure depuis le point de départ. Les valeurs possibles sont : `Clockwise` et `Counterclockwise`.
`IsLargeArc`	Indique si l'arc dépasse 180°.

Classe BezierSegment

Classe BezierSegment	
Attribut	**Utilité**
`Point1`	Définit le premier point de contrôle du segment
`Point2`	Définit le second point de contrôle du segment
`Point3`	Définit le troisième point de contrôle du segment

Classe Border

Classe Border	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`ContextMenu`	Menu contextuel associé à ce cadre
`CornerRadius`	Facteur d'arrondissement des coins
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.

Classe Border	
Attribut	**Utilité**
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Peut également s'appliquer aux éléments `ListBoxItem`.
`Margin`	Marges autour du cadre
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`Style`	Style à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Button

Classe Button	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`Content`	Texte affiché
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Height`	Hauteur

Classe Button	
Attribut	**Utilité**
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsCancel	Indique s'il s'agit du bouton associé à la touche Echap
IsDefault	Indique s'il s'agit ou non du bouton par défaut.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
TabIndex	Position dans l'ordre de déplacement avec la touche de tabulation
ToolTip	Info-bulle associée au bouton
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
VerticalContentAlignment	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe Canvas

Classe Canvas	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
Height	Hauteur

Classe Canvas	
Attribut	**Utilité**
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Canves: Attributs attachés	
Attribut attaché	**Utilité**
`Top`	Position par rapport au bord supérieur
`Left`	Position par rapport au bord gauche
`Bottom`	Position par rapport au bord inférieur
`Right`	Position par rapport au bord droit

Pour plus d'informations sur les attributs attachés reportez-vous page 57.

Classe CheckBox

Classe CheckBox	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`Content`	Texte affiché

Classe CheckBox	
Attribut	**Utilité**
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsChecked	Détermine si la case est cochée ou non.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False.
IsThreeState	Détermine s'il s'agit d'une case à cocher à 2 ou 3 états
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
VerticalContentAlignment	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe ColorAnimation

Classe ColorAnimation	
Attribut	**Utilité**
`AutoReverse`	Permet de réaliser automatiquement l'animation en sens inverse à la fin.
`Duration`	Durée de l'animation
`From`	Couleur initiale
`RepeatBehavior`	Permet la répétition automatique de l'animation. Il existe trois types de répétition : `IterationCount`, `RepeatDuration` et `Forever`.
`To`	Couleur finale

Classe ComboBox

Classe ComboBox	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`ComboBoxItem`	Valeurs contenues dans la `ListBox`. Chaque valeur est contenue dans un nœud enfant `ListBoxItem`.
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`HorizontalContentAlignment`	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEditable`	Détermine si le texte est éditable ou si la valeur doit être impérativement choisie dans la liste.

Classe ComboBox	
Attribut	**Utilité**
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Peut également s'appliquer aux éléments ComboBoxItem.
IsReadOnly	Met la zone d'encodage en lecture seule.
IsSelected	S'applique à ComboBoxItem. Détermine si la valeur est sélectionnée ou non.
IsTextSearchEnabled	Permet de désactiver ou de réactiver la possibilité de rechercher dans la liste.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
SelectedIndex	Détermine l'index de l'élément sélectionné.
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
VerticalContentAlignment	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe DiffuseMaterial

Classe DiffuseMaterial	
Attribut	**Utilité**
AmbientColor	Définit la couleur d'ambiance de la texture
Color	Définit la couleur de la texture
UpDirection	Définit le contenu de la texture

Classe DirectionalLight

Classe DirecionalLight	
Attribut	**Utilité**
Color	Couleur de la lumière
Direction	Orientation du spot

Classe DockPanel

Classe DockPanel	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe DockPanel: Attributs attachés	
Attribut attaché	**Utilité**
`Dock`	Indique où doit avoir lieu le docking. Les valeurs possibles sont `Top`, `Bottom`, `Left` et `Right`.

Classe DocumentViewer

Classe DocumentViewer	
Attribut	**Utilité**
`ShowPageBorders`	Détermine si les bords autour du document doivent être affichés
`VerticalPageSpacing`	Définit l'espace entre deux pages
`Zoom`	Définit le zoom

Classe DoubleAnimation

Classe DoubleAnimation	
Attribut	**Utilité**
`AutoReverse`	Permet de réaliser automatiquement l'animation en sens inverse à la fin.
`Duration`	Durée de l'animation
`From`	Valeur initiale
`RepeatBehavior`	Permet la répétition automatique de l'animation. Il existe trois types de répétition : `IterationCount`, `RepeatDuration` et `Forever`.
`To`	Valeur finale

Classe DoubleAnimationUsingKeyFrames

Classe DoubleAnimationUsingKeyFrames	
Attribut	**Utilité**
`AutoReverse`	Permet de réaliser automatiquement l'animation en sens inverse à la fin.
`BeginTime`	Pour postposer le début de l'animation
`Duration`	Durée de l'animation
`RepeatBehavior`	Permet la répétition automatique de l'animation. Il existe trois types de répétition : `IterationCount`, `RepeatDuration` et `Forever`.

Classe Ellipse

Classe Ellipse	
Attribut	**Utilité**
`Height`	Hauteur de l'ellipse
`Width`	Largeur de l'ellipse
`Fill`	Définit le fond de l'ellipse
`StrokeThickness`	Largeur du trait
`Stroke`	Définit le contenu du trait. Généralement la couleur.

L'ellipse est positionnée en utilisant les attributs attachés du conteneur. Exemple: `Canvas.Top`.

Classe EventTrigger

Classe EventTrigger	
Attribut	**Utilité**
`RoutedEvent`	Evénement qui déclenche le trigger

Classe Expander

Classe Expander	
Attribut	**Utilité**
`ExpandDirection`	Détermine le sens de l'expansion
`Header`	Titre
`IsEnabled`	Si cet attribut est `false`, l'`Expander` et son contenu (sauf mention contraire) est désactivé
`IsExpanded`	Détermine si le contenu est affiché ou caché
`Name`	Nom de l'instance

Classe Figure

Classe Figure	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Height`	Hauteur de la figure
`HorizontalAnchor`	Détermine la position horizontale de l'ancre. Les valeurs possibles sont `ContentRight`, `ContentLeft`, `ContentCenter`, `PageRight`, `PageLeft`, `PageCenter`, `ParagraphRight` et `ParagraphLeft`

Classe Figure	
Attribut	**Utilité**
`HorizontalOffset`	Déplacement horizontal de la figure par rapport à son ancre. La valeur peut être positive ou négative
`Margin`	Fixe les marges autour de la zone.
`Name`	Nom de l'instance
`TextAlignment`	Alignement du texte. Les valeurs possibles sont `Left`, `Rigth`, `Height`, `Justify`
`VerticalAnchor`	Détermine la position verticale de l'ancre. Les valeurs possibles sont `ContentTop`, `ContentBottom`, `ContentCenter`, `PageTop`, `PageBottom`, `PageCenter` et `ParagraphTop`
`VerticalOffset`	Déplacement vertical de la figure par rapport à son ancre. La valeur peut être positive ou négative
`Width`	Largeur de la figure

Classe FixedPage

Classe FixedPage	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`HorizontalAlignment`	Alignement horizontal de la page dans son contenant
`Margin`	Fixe les marges autour du texte.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum

Classe FixedPage	
Attribut	**Utilité**
MinWidth	Largeur minimum
Name	Nom de l'instance
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
VerticalAlignment	Alignement vertical de la page dans son contenant
Width	Largeur

Class FixedDocument

Classe FixedDocument	
Attribut	**Utilité**
Pages	Collection d'objets PageContent qui définit chaque page

Classe Floater

Classe Floater	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
BorderBrush	Couleur du bord.
BorderThickness	Epaisseur du bord
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
HorizontalAlignment	Alignement de la zone dans la ligne. Les valeurs possibles sont Left, Right et Center
Margin	Fixe les marges autour de la zone.
Name	Nom de l'instance
TextAlignment	Alignement du texte. Les valeurs possibles sont Left, Rigth, Height, Justify
Width	Largeur de la zone

Classe FlowDocument

Classe FlowDocument	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`MaxPageHeight`	Hauteur maximum d'une page
`MaxPageWidth`	Largeur maximum d'une page
`MinPageHeight`	Hauteur minimum d'une page
`MinPageWidth`	Largeur minimum d'une page
`Name`	Nom de l'instance
`PageHeight`	Hauteur d'une page
`PageWidth`	Largeur d'une page
`TextAlignment`	Alignement du texte. Les valeurs possibles sont `Left`, `Rigth`, `Height`, `Justify`

Classe GradientStop

Classe GradientStop	
Attribut	**Utilité**
`Color`	Couleur à utiliser pour le dégradé
`Offset`	Position relative ou prend effet la nouvelle couleur. Valeur entre 0 et 1

Classe Grid

Classe Grid	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`ColumnDefinitions`	Définition des colonnes. Contient des éléments `ColumnDefinition`.

Classe Grid	
Attribut	**Utilité**
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`RowDefinitions`	Définition des lignes. Contient des éléments `RowDefinition`.
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Grid: Attributs attachés	
Attribut attaché	**Utilité**
Row	Numéro de la ligne
Column	Numéro de la colonne
RowSpan	Nombre de lignes regroupées
ColumnSpan	Nombre de colonnes regroupées

Classe GridSplitter

Classe GridSplitter	
Attribut	**Utilité**
`Grid.Column`	Colonne où est placé le `GridSplitter`.
`Grid.ColumnSpan`	Nombre de colonnes sur lesquelles s'étant le `GridSplitter`.
`Grid.Row`	Ligne où est placé le `GridSplitter`.

Classe GridSplitter	
Attribut	**Utilité**
`Grid.RowSpan`	Nombre de lignes sur lesquelles s'étant le `GridSplitter`.
`HorizontalAlignment`	Alignement horizontal utilisé surtout avec `ResizeDirection="Rows"`
`ResizeDirection`	Définit le sens. Vertical si la valeur est `Columns` et horizontal pour l valeur `Rows`
`ShowPreview`	Détermine si le bord est directement déplacé ou si une prévisualisation est d'abord effectuée

Classe GridView

Classe GridView	
Attribut	**Utilité**
`AllowsColumnReorder`	Les valeurs possibles sont `True` et `False`. Autorise ou non l'utilisateur à déplacer les colonnes
`Columns`	Collection d'objet `GridViewColumn` définissant les colonnes du `GridView`

Classe GridViewColumn

Classe GridViewColumn	
Attribut	**Utilité**
`DisplayMemberBinding`	Définit la propriété de l'objet associé qui doit être affiché dans cette colonne.
`Header`	Titre de la colonne
`Width`	Largeur de la colonne

Classe Hyperlink

Classe Hyperlink	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères

Classe Hyperlink	
Attribut	**Utilité**
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
Name	Nom de l'instance
NavigateUri	Adresse de destination
Text	Texte affiché

Classe Image

Classe Image	
Attribut	**Utilité**
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
Source	Source de l'image
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe ImageBrush

Classe ImageBrush	
Attribut	**Utilité**
AlignmentX	Alignement horizontal. Les valeurs possibles sont : Left, Center et Right
AlignmentY	Alignement vertical. Les valeurs possibles sont : Top, Center et Bottom.

Classe ImageBrush	
Attribut	**Utilité**
ImageSource	Chemin et nom du fichier contenant l'image.
Stretch	Les valeurs possibles sont : None, Stretch, Uniform, UniformToFill
TileMode	Les valeurs possibles sont None, Tile, FlipX, FlipY et FlipXY
Viewport	Permet de contrôler la dimension relative de l'image en vue de réaliser une mosaïque

Classe Label

Classe Label	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
BorderBrush	Couleur du bord.
BorderThickness	Epaisseur du bord
Content	Texte affiché
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance

Classe Label	
Attribut	**Utilité**
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`VerticalContentAlignment`	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Line

Classe Line	
Attribut	**Utilité**
`X1`	Coordonnée horizontale du point de départ
`Y1`	Coordonnée verticale du point de départ
`X2`	Coordonnée horizontale du point d'arrivée
`Y2`	Coordonnée verticale du point d'arrivée
`StrokeThickness`	Largeur du trait
`Stroke`	Définit le contenu du trait. Généralement la couleur.

Classe LinearGradientBrush

Classe LinearGradientBrush	
Attribut	**Utilité**
`EndPoint`	Point ou s'arrête le dégradé
`GradientStop`	Collection des points de contrôle pour le dégradé. À chaque point de contrôle, vous pourrez changer les paramètres tels que la couleur
`StartPoint`	Point de départ pour le dégradé

Classe LineSegment

Classe LineSegment	
Attribut	**Utilité**
`Point`	Point d'arrivée

Classe ListBox

Classe ListBox	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
BorderBrush	Couleur du bord.
BorderThickness	Epaisseur du bord
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Peut également s'appliquer aux éléments ListBoxItem.
IsSelected	S'applique à ListBoxItem. Détermine si la valeur est sélectionnée ou non.
ListBoxItem	Valeurs contenues dans la ListBox. Chaque valeur est contenue dans un nœud enfant ListBoxItem.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
SelectionMode	Détermine le mode de sélection. Les valeurs possibles sont Single, Multiple, Extend
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.

Classe ListBox	
Attribut	**Utilité**
`VerticalContentAlignment`	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe ListView

Classe ListView	
Attribut	**Utilité**
`Height`	Hauteur de la `ListView`
`ItemSource`	Source des données inclues dans la `ListView`
`ItemTemplate`	Définit la présentation des données en utilisant un `DataTemplate`
`Margin`	Marge autour de la `ListView`
`Name`	Nom de l'instance
`VerticalAlignment`	Alignement du contenu

Classe Menu

Classe Menu	
Attribut	**Utilité**
`Height`	Hauteur de la barre de menu
`Items`	Collection d'objet `MenuItem` qui détermine les éléments du menu
`Name`	Nom de l'instance
`VerticalAlignment`	Alignement de la barre de menu dans son conteneur

Classe MenuItem

Classe MenuItem	
Attribut	**Utilité**
`Header`	Texte affiché
`IsChecked`	Définit si ce point du menu est coché ou non

Classe MenuItem	
Attribut	**Utilité**
IsEnable	Définit si ce point du menu est actif ou non
Name	Nom de l'instance

Classe MeshGeometry3D

Classe MeshGeometry3D	
Attribut	**Utilité**
Positions	Collection de vertex position pour tracer la forme

Classe NavigationWindow

Classe NavigationWindow	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
CanGoBack	Le retour arrière est possible
CanGoForward	Le renvoi vers l'avant est possible
DataContext	Contexte pour la liaison aux données
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Les contrôles enfants sont également désactivés.
Margin	Marges autour du contrôle
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum

Classe NavigationWindow	
Attribut	**Utilité**
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`NavigationService`	Offre des services pour naviguer entre les pages
`RenderTransform`	Transformation à appliquer
`Ressources`	Collection des ressources utilisables dans la page
`Source`	Adresse URI de la page à afficher dans la fenêtre
`Title`	Titre de la page
`Triggers`	Collection de *trigger* associés à la fenêtre
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur
`WindowStartupLocation`	Position de la fenêtre

Classe ObjectDataProvider

Classe ObjectDataProvider	
Attribut	**Utilité**
`ConstructorParameters`	Paramètres à transmettre au constructeur
`MethodName`	Nom doit être appelée de la méthode qu
`MethodParameters`	Liste des paramètres qui doivent être transmis à la méthode
`ObjectType`	Type de l'objet qui doit être créé
`ObjectInstance`	Instance utilisée comme source

Classe Page

Classe Page	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`DataContext`	Contexte pour la liaison aux données
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage

Classe Page	
Attribut	**Utilité**
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Les contrôles enfants sont également désactivés.
Margin	Marges autour du contrôle
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
NavigationService	Offre des services pour naviguer entre les pages
RenderTransform	Transformation à appliquer
Ressources	Collection des ressources utilisables dans la page
Title	Titre de la page
Triggers	Collection de trigger associés à la page
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur
WindowHeight	Hauteur de la fenêtre
WindowTitle	Titre de la fenêtre
WindowWidth	Largeur de la fenêtre

Classe PageContent

Classe PageContent	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
DataContext	Contexte pour la liaison aux données
Height	Hauteur

Classe PageContent	
Attribut	**Utilité**
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Les contrôles enfants sont également désactivés.
Margin	Marges autour du contrôle
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
Ressources	Collection des ressources utilisables dans la page
Source	Fichier xaml à charger dans le contenu de la page
Title	Titre de la page
Triggers	Collection de trigger associés au contrôle
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe Paragraph

Classe Paragraph	
Attribut	**Utilité**
Background	Détermine la couleur mais aussi le contenu du fond
BreakPageBefore	Demande un saut de page avant le paragraphe
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
KeepTogether	Garder le paragraphe sur une même page

Classe Paragraph	
Attribut	**Utilité**
KeepWithNext	Garder le paragraphe sur la même page que le suivant
Margin	Marges autour du paragraphe
Name	Nom du paragraphe
Text	Texte contenu dans le paragraphe
TextAlignment	Alignement du texte. Les valeurs possibles sont Left, Rigth, Height, Justify
TextIndent	Indentation de la première ligne

Classe Path

Classe Path	
Attribut	**Utilité**
StrokeThickness	Largeur du trait
Stroke	Définit le contenu du trait. Généralement la couleur.
Fill	Définit le remplissage
Data	Contient un objet de type Geometry qui représente une forme complexe réalisée avec différentes figures.

Classe PathFigure

Classe PathFigure	
Attribut	**Utilité**
StartPoint	Point de départ de la figure
Segments	Collection de segments successifs pour utilisés pour former la figure. Les segments peuvent être de type ArcSegment, BezierSegment, LineSegment, PolyBezierSegment, PolyLineSegment, PolyQuadraticBezierSegment, QuadraticBezierSegment.

Classe Pen

Classe Pen	
Attribut	**Utilité**
Thickness	Largeur du trait
DashStyle	Style du trait (continu, pointillé, ...)

Classe Pen	
Attribut	**Utilité**
`Brush`	Définit le contenu du trait
`Color`	Définit la couleur du trait

Classe PerspectiveCamera

Classe PerspectiveCamera	
Attribut	**Utilité**
`LookDirection`	Participe à déterminer l'angle de vue en indiquant a direction
`Position`	Définit la position de la caméra dans l'espace 3D
`UpDirection`	Participe à déterminer l'angle de vue en indiquant a direction

Classe Polygon

Classe Polygon	
Attribut	**Utilité**
`Points`	Collection des points déterminant les sommets
`StrokeThickness`	Largeur du trait
`Stroke`	Définit le contenu du trait. Généralement la couleur.
`Fill`	Définit le remplissage du polygone

Classe Polyline

Classe Polyline	
Attribut	**Utilité**
`Points`	Collection des points de passage relié par des lignes
`StrokeThickness`	Largeur du trait
`Stroke`	Définit le contenu du trait. Généralement la couleur.

Casse PolylineSegment

Casse PolylineSegment	
Attribut	**Utilité**
`Points`	Définit les points de passage

Classe Popup

Classe Popup	
Attribut	**Utilité**
`Name`	Nom de l'instance
`PlacementRectangle`	Taille du popup
`PlacementTarget`	Elément auquel le popup est attaché

Classe RadialGradientBrush

Classe RadialGradientBrush	
Attribut	**Utilité**
`GradientStops`	Collection des points de contrôle pour le dégradé. À chaque point de contrôle, vous pourrez changer les paramètres tels que la couleur

Classe RadioButton

Classe RadioButton	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.
`BorderThickness`	Epaisseur du bord
`Content`	Texte affiché
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`GroupName`	Nom du groupe dans lequel est inclus ce `RadioButton`
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.

Classe RadioButton	
Attribut	**Utilité**
`HorizontalContentAlignment`	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsChecked`	Détermine si la case est cochée ou non.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`VerticalContentAlignment`	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Rectangle

Classe Rectangle	
Attribut	**Utilité**
`Height`	Hauteur de l'ellipse
`Width`	Largeur de l'ellipse
`RadiusX`	Radiant en X utilisé pour arrondir les coins
`RadiusY`	Radiant en X utilisé pour arrondir les coins
`Fill`	Définit le fond de l'ellipse
`StrokeThickness`	Largeur du trait
`Stroke`	Définit le contenu du trait. Généralement la couleur.

Le rectangle est positionné en utilisant les attributs attachés du conteneur. Exemple: `Canvas.Top`.

Classe RotateTransform

Classe RotateTransform	
Attribut	**Utilité**
`Angle`	Détermine l'angle de rotation en degré
`CenterX`	Modifie la coordonnée X du point centrale de la rotation
`CenterY`	Modifie la coordonnée Y du point centrale de la rotation

Classe RepeatButton

Voir aussi la classe `Button` ***page 368.***

Classe RepeatButton	
Attribut	**Utilité**
`Cursor`	Type de curseur pour la souris
`Delay`	Délai avant de commencer les répétitions
`Interval`	Intervalle entre deux appels à l'événement `Click`.

Classe ScaleTransform

Classe ScaleTransform	
Attribut	**Utilité**
`CenterX`	Déplace horizontalement l'objet
`CenterY`	Déplace verticalement l'objet
`ScaleX`	Détermine le facteur de multiplication horizontal
`ScaleY`	Détermine le facteur de multiplication vertical

Classe ScrollViewer

Classe ScrollViewer	
Attribut	**Utilité**
`Name`	Nom de l'instance
`HorizontalScrollBar Visibility`	Détermine comment la barre de défilement doit être gérée. Les valeurs possibles sont `Disabled`, `Visible`, `Hidden` et `Auto`.

Classe ScrollViewer	
Attribut	**Utilité**
`VerticalScrollBar Visibility`	Détermine comment la barre de défilement doit être gérée. Les valeurs possibles sont `Disabled`, `Visible`, `Hidden` et `Auto`.

Classe Section

Classe Section	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BreakPageBefore`	Demande un saut de page avant le paragraphe
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Margin`	Marges autour du paragraphe
`Name`	Nom de la section
`TextAlignment`	Alignement du texte. Les valeurs possibles sont `Left`, `Rigth`, `Height`, `Justify`

Classe Setter

Classe Setter	
Attribut	**Utilité**
`Property`	Propriété à atteindre
`Value`	Valeur à assigner à la propriété

Classe SkewTransform

Classe SkewTransform	
Attribut	**Utilité**
`AngleX`	Angle de rotation sur l'axe horizontal
`AngleY`	Angle de rotation sur l'axe vertical

Classe SkewTransform	
Attribut	**Utilité**
CenterX	Détermine la position centrale horizontale de la transformation
CenterY	Détermine la position centrale verticale de la transformation

Classe Slider

Classe Slider	
Attribut	**Utilité**
AutoToolTipPlacement	Position du tooltip automatique
AutoToolTipPrecision	Nombre de décimales affichées dans le tooltip automatique
Background	Détermine la couleur mais aussi le contenu du fond
BorderBrush	Couleur du bord.
BorderThickness	Epaisseur du bord
Cursor	Type de curseur pour la souris
Delay	Délai avant de commencer les répétitions
Foreground	Couleur du texte
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
Interval	Intervalle entre deux incrémentations.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Peut également s'appliquer aux éléments ListBoxItem.
IsSelectionRangeEnabled	Permet l'affichage d'une zone
IsSnapToTickEnabled	Détermine si la valeur sélectionnée est automatiquement ajustée à un repère
LargeChange	Incrément lors du changement de valeur via un click sur le contrôle
MaxHeight	Hauteur maximum
Maximum	Valeur maximale
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
Minimum	Valeur minimale

Classe Slider	
Attribut	**Utilité**
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`Orientation`	Détermine l'orientation horizontale ou verticale
`RenderTransform`	Transformation à appliquer
`SelectionEnd`	Indique la valeur maximale de la zone affichée
`SelectionStart`	Indique la valeur minimale de la zone affichée
`TickFrequency`	Espacement entre les repères
`TickPlacement`	Position des repères visuels
`Ticks`	Liste des valeurs pour les repères visuels
`Value`	Valeur indiquée par le curseur
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe SolidColorBrush

Classe SolidColorBrush	
Attribut	**Utilité**
`Color`	Détermine la Détermine la couleur mais aussi le contenu du fond
`Transform`	Définit la transformation à appliquer

Classe SplineDoubleKeyFrame

Classe SplineDoubleKeyFrame	
Attribut	**Utilité**
`KeyTime`	Temps où elle doit être atteinte
`Value`	Valeur à atteindre

Classe StackPanel

Classe StackPanel	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond

Classe StackPanel	
Attribut	**Utilité**
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe StoryBoard

Classe StoryBoard	
Attribut attaché	**Utilité**
`TargetName`	Nom de l'objet qui est animé
`TargetProperty`	Nom de la propriété qui subit l'animation

Classe Style

Classe Style	
Attribut	**Utilité**
`X:Key`	Nom du style. Ce nom est utilisé par les contrôles souhaitant appliquer le style. Il s'agit de la méthode de nommage utilisée pour les ressources.
`TargetType`	Type de contrôle. Le style est automatiquement appliqué à tous les contrôles de ce type.

Classe Table

Classe Table	
Attribut	**Utilité**
`BorderBrush`	Couleur du bord
`BorderThickness`	Largeur du bord
`Columns`	Collection d'objets `TableColumn` qui définissent les colonnes
`RowGroups`	Contient un objet de type `TableRowGroup` et qui contient les lignes de la table

Classe TableCell

Classe TableCell	
Attribut	**Utilité**
`Block.TextAlignment`	Alignement du texte dans la cellule
`ColumnSpan`	Nombre de colonnes sur lesquels s'étend cette cellule

Classe TableColumn

Classe TableColumn	
Attribut	**Utilité**
`Background`	Définit le fond de la colonne
`Width`	Définit la largeur de la colonne

Classe TableRow

Classe TableRow	
Attribut	**Utilité**
`Background`	Définit le fond de la ligne

Classe TabControl

Classe TabControl	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`BorderBrush`	Couleur du bord.

Classe TabControl	
Attribut	**Utilité**
BorderThickness	Epaisseur du bord
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False. Les contrôles enfants sont également désactivés.
MaxHeight	Hauteur maximum
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
RenderTransform	Transformation à appliquer
TabItem	Le TabControl contient un ensemble de TabItem
TabItem.Header	Header ne fait pas partie de la classe TabControl mais bien de la classe TabItem. Il représente le texte affiché dans l'onglet.
VerticalAlignment	Alignement vertical du contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
VerticalContentAlignment	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont Top, Bottom, Center, Stretch.
Width	Largeur

Classe TabItem

Classe TabIem	
Attribut	**Utilité**
Header	Texte contenu dans l'onglet

Classe TextBlock

Classe TextBox	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Foreground`	Couleur du texte
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`Margin`	Fixe les marges autour du texte.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`Text`	Texte affiché
`TextAlignment`	Alignement du texte. Les valeurs possibles sont `Left`, `Rigth`, `Height`, `Justify`
`TextTrimming`	Gère le comportement en bout de contrôle. Les valeurs possibles sont `None`, `WordEllipsis`, `CharacterEllipsis`
`TextWrapping`	Gère le comportement en bout de ligne. Les valeurs possibles sont `NoWrap`, `Wrap` et `WrapWithOverflow`.
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe TextBox

Classe TextBox	
Attribut	**Utilité**
AcceptsReturn	Autorise le retour à la ligne imposé dans le contenu. Doit être True ou False
AcceptsTab	Autorise l'utilisation des tabulations dans le contenu. Doit être True ou False.
Background	Détermine la couleur mais aussi le contenu du fond
BorderBrush	Couleur du bord.
BorderThickness	Epaisseur du bord
CharacterCasing	Case du contenu. Les valeurs possibles sont Upper, Lower
FontFamily	Police d'affichage
FontSize	Taille de la police d'affichage
FontStretch	Espacement des caractères
FontStyle	Style de la police. Italic, Normal, Oblique
FontWeight	Epaisseur des traits
Foreground	Couleur du texte
Height	Hauteur
HorizontalAlignment	Alignement horizontal du contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
HorizontalContentAlignment	Alignement horizontal du texte dans le contrôle. Les valeurs possibles sont Left, Rigth, Center, Stretch.
IsEnabled	Détermine si le contrôle est actif ou non. Les valeurs possibles sont True ou False.
IsReadOnly	Détermine si le contrôle est en lecture seule ou non. Les valeurs possibles sont True ou False.
MaxHeight	Hauteur maximum
MaxLength	Nombre maximum de caractères
MaxLines	Nombre de lignes maximal affichées
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinLines	Nombre de lignes minimal affichées
MinWidth	Largeur minimum

Classe TextBox	
Attribut	**Utilité**
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`Text`	Texte affiché
`TextWrapping`	Détermine le comportement dans le cas où le contenu atteint l'extrémité du contrôle. Les valeurs possibles sont `NoWrap`, `Wrap`, `WrapWthOverflow`
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`VerticalContentAlignment`	Alignement vertical du texte dans le contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe Toolbar

Classe Toolbar	
Attribut	**Utilité**
`Band`	Détermine dans quelle bande de la `ToolBarTray` doit être positionné la `ToolBar`
`BandIndex`	Détermine l'ordre de la `ToolBar` dans la bande
`Height`	Hauteur de la barre d'outils
`Width`	Largeur de la barre d'outils

Classe ToolbarTray

Classe ToolbarTray	
Attribut	**Utilité**
`IsLocked`	Autorise ou non le déplacement des barres d'outils au sein de la `ToolBarTray`
`Orientation`	Détermine l'orientation. Les valeurs possibles sont `Vertical` et `Horizontal`
`Background`	Définit le fond

Classe TranslateTransform

Classe TranslateTransform	
Attribut	**Utilité**
X	Déplacement sur l'axe horizontal
Y	Déplacement sur l'axe vertical

Classe TreeView

Classe TreeView	
Attribut	**Utilité**
Background	Défini le fond du contrôle. Typiquement la couleur.
Height	Hauteur
Items	Collection de TreeViewItems contenant la hiérarchie des nœuds contenu dans le TreeView.
Margin	Défini les marges autour du contrôle
MaxHeight	Hauteur maximum du
MaxWidth	Largeur maximum
MinHeight	Hauteur minimum
MinWidth	Largeur minimum
Name	Nom de l'instance
Width	Largeur

Classe TreeViewItem

Classe TreeViewItem	
Attribut	**Utilité**
Header	Défini le titre du noeud
ItemsSource	Défini la source des données contenues dans le noeud.
ItemTemplate	Défini la forme des données affichées.

Classe Trigger

Classe Trigger	
Attribut	**Utilité**
Property	Nom de la propriété qui va déclencher le trigger

Classe Trigger	
Attribut	**Utilité**
`Setters`	Collection d'objets `Setter` qui seront appliqués lorsque le trigger est activé.
`Value`	Valeur de la propriété définie avec l'attribut `Property` et qui déclenche le *trigger*.

Classe ViewBox

Classe ViewBox	
Attribut	**Utilité**
`Height`	Hauteur
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`Width`	Largeur
`Stretch`	Défini la manière dont le contenu sera étendu. Les valeurs possibles sont : `Fill`, `None`, `Uniform`, `UniformToFill`
`StretchDirection`	Détermine le sens de l'étirement. Les valeurs possibles sont : `Both`, `DownOnly`, `UpOnly`

Classe Viewport3D

Classe Viewport3D	
Attribut	**Utilité**
`Camera`	Contient un objet de type `Camera` qui gère la projection du contenu en 3 dimensions
`Height`	Hauteur
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`Width`	Largeur

Classe Window

Classe Window	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`DataContext`	Contexte pour la liaison aux données
`FontFamily`	Police d'affichage
`FontSize`	Taille de la police d'affichage
`FontStretch`	Espacement des caractères
`FontStyle`	Style de la police. `Italic`, `Normal`, `Oblique`
`FontWeight`	Epaisseur des traits
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`Margin`	Marges autour du contrôle
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`Ressources`	Collection des ressources utilisables dans la page
`Title`	Titre de la page
`Triggers`	Collection de *triggers* associés à la fenêtre
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur
`WindowStartupLocation`	Position de la fenêtre

Classe WrapPanel

Classe WrapPanel	
Attribut	**Utilité**
`Background`	Détermine la couleur mais aussi le contenu du fond
`Height`	Hauteur
`HorizontalAlignment`	Alignement horizontal du contrôle. Les valeurs possibles sont `Left`, `Rigth`, `Center`, `Stretch`.
`IsEnabled`	Détermine si le contrôle est actif ou non. Les valeurs possibles sont `True` ou `False`. Les contrôles enfants sont également désactivés.
`MaxHeight`	Hauteur maximum
`MaxWidth`	Largeur maximum
`MinHeight`	Hauteur minimum
`MinWidth`	Largeur minimum
`Name`	Nom de l'instance
`RenderTransform`	Transformation à appliquer
`VerticalAlignment`	Alignement vertical du contrôle. Les valeurs possibles sont `Top`, `Bottom`, `Center`, `Stretch`.
`Width`	Largeur

Classe XmlDataProvider

Classe XmlDataProvider	
Attribut	**Utilité**
`X:Key`	Nom de l'instance
`Source`	Fichier XML contenant les données

13.5 Classes autorisées dans la zone internet

Pour le développement d'application hébergée sur un site Internet, il est utile de connaître les classes autorisées à être exécutées sans extension des droits.

Layout

- Canvas
- DockPanel
- Grid
- StackPanel
- TextBlock
- Viewbox
- FlowDocument

Controls

- Border
- Button
- ComboBox
- Frame
- Hyperlink
- Label
- Menu
- Page
- Popup. (Limité à la zone d'affichage du navigateur.)
- ScrollBar
- ScrollViewer
- TextBox
- Thumb
- ToolTip

Graphics and Animation

- Les classes de dessin 2D et 3D
- Les classes gérant les animations
- MediaElement (audio et vidéo)
- Glyphs
- Path
- Image

Autres

- Les classes assurant le *data binding*
- Les classes assurant la navigation
- Les classes relatives aux styles

13.6 Liste des touches de raccourcis pour les commandes d'édition

Touches de raccourci pour les commandes d'édition	
Commande	**Raccourci**
AlignCenter	Ctrl+E
AlignJustify	Ctrl+J
AlignLeft	Ctrl+L
AlignRight	Ctrl+R
Backspace	Retour arrière
CorrectSpellingError	-
DecreaseFontSize	Ctrl+
DecreaseIndentation	Ctrl+Maj+T
Delete	Delete
DeleteNextWord	Ctrl+Suppr
DeletePreviousWord	Ctrl+ Retour arrière
EnterLineBreak	Shift+Entrée
EnterParagraphBreak	Entrée
IgnoreSpellingError	-
IncreaseFontSize	Ctrl+OemCloseBrackets
IncreaseIndentation	Ctrl+T
MoveDownByLine	Bas
MoveDownByPage	Page suivante
MoveDownByParagraph	Ctrl+Bas
MoveLeftByCharacter	Gauche
MoveLeftByWord	Ctrl+Gauche
MoveRightByCharacter	Droite
MoveRightByWord	Ctrl+Droite
MoveToDocumentEnd	Ctrl+↘

Touches de raccourci pour les commandes d'édition	
Commande	**Raccourci**
MoveToDocumentStart	Ctrl+Début
MoveToLineEnd	↘
MoveToLineStart	Début
MoveUpByLine	Haut
MoveUpByPage	Page précédente
MoveUpByParagraph	Ctrl+Haut
SelectDownByLine	Maj+Bas
SelectDownByPage	Maj+ Page précédente
SelectDownByParagraph	Ctrl+Maj+Bas
SelectLeftByCharacter	Maj+Gauche
SelectLeftByWord	Ctrl+ Maj+Gauche
SelectRightByCharacter	Maj+Droite
SelectRightByWord	Ctrl+ Maj+Droite
SelectToDocumentEnd	Ctrl+ Maj+↘
SelectToDocumentStart	Ctrl+ Maj+Home
SelectToLineEnd	Maj+↘
SelectToLineStart	Maj+Début
SelectUpByLine	Maj +Haut
SelectUpByPage	Maj + Page précédente
SelectUpByParagraph	Ctrl+ Maj +Haut
TabBackward	Maj +Tab
TabForward	Tab
ToggleBold	Ctrl+B
ToggleBullets	Ctrl+ Maj+L
ToggleInsert	Insérer
ToggleItalic	Ctrl+I
ToggleNumbering	Ctrl+ Maj+N
ToggleSubscript	Ctrl+OemPlus
ToggleSuperscript	Ctrl+ Maj+OemPlus
ToggleUnderline	Ctrl+U

13.7 Liste des classes par catégories

Présentation

- Border
- BulletDecorator
- Canvas
- DockPanel
- GridView
- GridSplitter
- GridView
- GroupBox
- Panel
- Separator
- StackPanel
- WrapPanel
- Viewbox

Boutons

- Button
- RadioButton
- RepeatButton

Menus

- ContextMenu
- Menu
- ToolBar

Listes de choix

- ComboBox
- ListBox
- TreeView

Selection de valeurs

- CheckBox
- Slider

Navigation

- Frame
- ScrollBar
- ScrollViewer
- TabControl

Informations utilisateurs

- Expander
- Label
- Popup
- ProgressBar
- StatusBar
- ToolTip

Documents

- DocumentViewer
- FlowDocumentPageViewer
- FlowDocumentReader
- FlowDocumentScrollViewer

Edition de texte

- TextBox
- RichTextBox
- PasswordBox

Multimédia

- Image
- MediaElement

13.8 Liste des couleurs prédéfinies

Couleurs prédéfinies
Couleur
AliceBlue
AntiqueWhite
Aqua
Aquamarine
Azure
Beige
Bisque
Black
BlanchedAlmond
Blue
BlueViolet
Brown
BurlyWood
CadetBlue
Chartreuse
Chocolate
Coral
CornflowerBlue
Cornsilk
Crimson
Cyan
DarkBlue
DarkCyan
DarkGoldenrod
DarkGray
DarkGreen
DarkKhaki
DarkMagenta
DarkOliveGreen
DarkOrange
DarkOrchid

Couleurs prédéfinies
Couleur
DarkRed
DarkSalmon
DarkSeaGreen
DarkSlateBlue
DarkSlateGray
DarkTurquoise
DarkViolet
DeepPink
DeepSkyBlue
DimGray
DodgerBlue
Firebrick
FloralWhite
ForestGreen
Fuchsia
Gainsboro
GhostWhite
Gold
Goldenrod
Gray
Green
GreenYellow
Honeydew
HotPink
IndianRed
Indigo
Ivory
Khaki
Lavender
LavenderBlush
LawnGreen
LemonChiffon

Couleurs prédéfinies
Couleur
LightBlue
LightCoral
LightCyan
LightGoldenrodYellow
LightGray
LightGreen
LightPink
LightSalmon
LightSeaGreen
LightSkyBlue
LightSlateGray
LightSteelBlue
LightYellow
Lime
LimeGreen
Linen
Magenta
Maroon
MediumAquamarine
MediumBlue
MediumOrchid
MediumPurple
MediumSeaGreen
MediumSlateBlue
MediumSpringGreen
MediumTurquoise
MediumVioletRed
MidnightBlue
MintCream
MistyRose
Moccasin
Name

Couleurs prédéfinies
Couleur
NavajoWhite
Navy
OldLace
Olive
OliveDrab
Orange
OrangeRed
Orchid
PaleGoldenrod
PaleGreen
PaleTurquoise
PaleVioletRed
PapayaWhip
PeachPuff
Peru
Pink
Plum
PowderBlue
Purple
Red
RosyBrown
RoyalBlue
SaddleBrown
Salmon
SandyBrown
SeaGreen
SeaShell
Sienna
Silver
SkyBlue
SlateBlue
SlateGray

Couleurs prédéfinies
Couleur
Snow
SpringGreen
SteelBlue
Tan
Teal
Thistle
Tomato
Transparent
Turquoise
Violet
Wheat
White
WhiteSmoke
Yellow
YellowGreen

Index

A

AcceptsReturn, 40
AcceptsTab, 40
Alignement, 25
    Horizontal, 26
    Vertical, 26
AlignmentX, 52
AlignmentY, 52
Angle, 220
AngleX, 221
AngleY, 221
Animation, 241, 346
Annotation, 282
    AnnotationService, 283
Application, 133
ApplicationDefinition, 134
ArcSegment, 322, 368
Attribut, 14
Aurora Designer, 310

B

Background, 28, 342
Balise, 14
Barre d'outils, 183
    Contrôle, 184
    Débordement, 188
    Flottante, 188
    Orientation, 187
Barre de statut, 281
Barres de défilement, 77
BeginStoryboard, 242
BeginTime, 246
BezierSegment, 322, 368
Binding (voir Liaison), 195
Blocks, 280
Bold, 34
Bord, 47
    Mobile, 66
Border, 47, 368
BorderBrush, 23
BorderThickness, 23
Bouton, 46
    Défaut, 47
    Délai, 112
    Echap, 47
    Intervalle, 112
    Répétition, 112
Bouton radio (voir RadioButton), 102
BreakPageAfter, 256
BreakPageBefore, 256
Brush, 243
Bulle d'information, 106
Bulle Info-bulle, 106
Bullet, 260
BulletDecorator, 259
Button, 369
ByteAnimation, 241

C

Cadre, 47
    Arrondir, 48
CanGoBack, 149
CanGoForward, 149
Canvas, 56, 370
    Bottom, 59
    Left, 57
    Right, 59
    Top, 57
Case à cocher (voir CheckBox), 100
CenterY, 221
Cercle, 318
CharacterCasing, 40
CheckBox, 100, 371
    Cochée, 101
    État indéterminé, 101
CIDER, 290
Click, 139

Collection, 15
ColorAnimation, 243, 373
ColSpan, 66
ColumnDefinitions, 62
Columns, 273
ColumnSpan, 63
ComboBox, 98, 373
    Éditable, 99
ComboBoxItem, 98
Connexion, 193
ConstructorParameters, 216
Content, 23
ContentElement, 366
ContentPresenter, 234
ContentSite, 234
Control, 364
ControlTemplate, 234
Coordonnées, 56
    Système de coordonnées, 60
CornerRadius, 48
Couleur, 28

D

DataContext, 194, 207, 209
DataSet, 192
    Liaison, 194
DataTable, 201
DataTemplate, 209, 214
Déboguer, 298
Debug, 298
Décoration, 35
Défilement, 77
Delay, 112, 115
Désactivé, 58
DiffuseMaterial, 374
DirectionalLight, 324, 374
DisplayMemberPath, 202, 234
Dock, 74
DockPanel, 72, 375
    Dock, 74
Document
    Charger, 279
    Imprimer, 280
    Sauver, 279
DocumentViewer, 254, 375
DoubleAnimation, 244, 376
DoubleAnimationUsingKeyFrames, 245, 376
DynamicResource, 225

E

Echelle, 221
Ellipse, 317, 376
EllipseGeometry, 323
EndPoint, 230
Evénement, 139
EventTrigger, 245, 348, 377
ExpandDirection, 120
Expander, 118, 377
Express Application, 142

F

Fenêtre navigable, 157
Figure, 270, 377
Fill, 317
FixedDocument, 250, 379
FixedPage, 378
Floater, 270, 379
FlowDocument, 254, 380
FlowDocumentPageViewer, 264
FlowDocumentReader, 264
FlowDocumentScrollViewer, 261
Focusable, 235
Font, 27
FontFamily, 27
FontSize, 27

FontStrech, 28
FontWeight, 28
Foreground, 28
Frame, 88, 153
Framework 3.0, 13
Freezable, 366
Fusion
    De colonnes, 63
    De lignes, 63

G

GoBack, 149, 161
GoForward, 149
Gradient
    Linéaire, 230
    Radial, 230
GradientStop, 229, 380
GradientStops, 230
Graphic Designer, 303
Grid, 61, 380
    Column, 62
    ColumnSpan, 63
    Row, 62
GridSplitter, 66, 381
GridView, 211, 382
GridViewColumn, 382
Grille, 61
    Bord mobile, 66
    Fusion de colonnes, 63
    Fusion de lignes, 63
GroupBox, 105
    Header, 106
GroupName, 103

H

Header, 106, 120
Height, 24, 49
HorizontalAlignment, 26
HorizontalAnchor, 268
HorizontalContentAlignment, 25
HorizontalScrollBarVisibility, 43, 78
HTML, 166
Hyperlink, 382

I

IIS, 141
Image, 48, 383
    Fond, 50
ImageBrush, 51, 383
ImageSource, 49
Imprimer, 280
    XAMLPad, 18
Inclinaison, 221
Info-bulle, 106
Int32Animation, 241
IntelliSense, 19
Interactive Designer, 306
Interval, 112, 115
IsCancel, 47
IsChecked, 101-102, 173
IsDefault, 47
IsEditable, 99
IsEnabled, 43, 58, 65, 95, 172
IsExpanded, 120
IsFocused, 241
IsMouseOver, 241
IsReadOnly, 43, 95
IsSelected, 94, 111
IsSnapToTickEnabled, 115
IsTextSearchEnabled, 100
IsThreeState, 101
Italic, 34
ItemSource, 209
ItemsPresenter, 235, 237
ItemTemplate, 209

J

JavaScript, 166

K

KeyTime, 246

L

Label, 22-23, 384
    Bord, 23
    Content, 23
LargeChange, 115
Liaison
    DataSet, 194
    Objet business, 203
Line, 316, 385
LinearGradientBrush, 230, 385
LineBreak, 38
LineGeometry, 323
LineSegment, 322, 385
List, 257
ListBox, 92, 386
    Mode de sélection, 94
    Valeur par défaut, 92, 94-95
ListBoxItem, 92
Liste, 92
ListView, 208, 387
    DataTemplate, 209
    ItemTemplate, 209
Loaded, 194, 242
LoadedBehavior, 126
Location, 35
LookDirection, 324

M

Majuscule imposée, 40
Marge, 33
Margin, 33
MarkerOffset, 259
MarkerStyle, 258
Matrice, 222
MaxHeight, 25, 64
MaxLength, 40
MaxLines, 42
MaxWidth, 25, 92
MediaElement, 126
MediaTimeLine, 127
Menu, 170, 387
    Action, 173
    Contextuel, 178, 226
    Dynamique, 176
    Inactif, 172
    Séparateur, 172
Menu contextuel
    Partage, 226
MenuItem, 170, 387
MeshGeometry3D, 324, 388
MethodName, 217
MethodParameters, 217
MinHeight, 25
MinLines, 42
Minuscule imposée, 40
MinWidth, 25
Mosaïque, 51
Mot de passe, 46
MouseMove, 246

N

Name, 29
Namespace, 151
    Namespaces, 16
Navigate, 148

Navigateur Internet, 140, 142
NavigationService, 147
NavigationWindow, 157-158, 388
Nœud XML, 14
Nom d'un contrôle, 29
Note, 283

O

ObjectDataProvider, 207, 214, 389
ObjectInstance, 217, 389
ObjectType, 216
Offset, 229
Onglet, 109
OnReturn, 149
Ordre, 88
    Des tabulations, 88
Orientation, 187

P

Page, 135, 143, 389
    Statique, 146
PageContent, 250, 390
PageFunction, 151
Paragraph, 254, 391
Parent, 164
PART_ContentHost, 233
Password, 46
PasswordBox, 46
Path, 321, 392
PathFigure, 320, 392
PathGeometry, 323
Pen, 35, 392
PerspectiveCamera, 393
Photo, 48
PlacementRectangle, 123
PlacementTarget, 123
Point3D, 324
Police (voir Font), 27
    XAMLPad, 18
PolyBezierSegment, 322
Polygon, 318, 393
Polyline, 318, 393
PolyLineSegment, 322, 393
PolyQuadraticBezierSegment, 322
Popup, 123, 394
Position, 324
PrespectiveCamera, 324
PrintDialog, 279
Projet, 132
Property, 227, 239
Propriété attachée, 57

Q

QuadraticBezierSegment, 322

R

RadialGradientBrush, 230, 237, 394
RadioButton, 102, 394
    Groupe, 103
    Sélection, 102
RadioButtonList, 103
Rectangle, 318, 395
RectangleGeometry, 323
RenderTransform, 220
RepeatButton, 112, 396
ResizeDirection, 66
ResizeMode, 136
Ressource, 223
    Dynamique, 225
    Nom, 224
    Statique, 225
    Utilisation, 224
RichTextBox, 275
Root, 16

RotateTransform, 220, 396
Rotation, 220
RoutedEvent, 245, 348
RowDefinitions, 62
RowSpan, 66

S

SartPoint, 230, 320
ScaleTransform, 221, 245, 396
ScaleX, 221
ScaleY, 221
ScrollViewer, 77, 396
Section, 257, 397
Sécurité, 141
SelectedIndex, 92
SelectedValue, 234
SelectedValuePath, 234
SelectionMode, 94
Setter, 227, 397
ShowInTaskbar, 138
ShowPageBorders, 254
ShowsNavigationUI, 148
ShowsPreview, 67
Size, 321
SizeToContent, 136
SkewTransform, 221, 397
Slider, 114, 398
SolidColorBrush, 224, 244, 399
SoundPlayerAction, 349
Source, 209, 252
    De données, 192
SplineDoubleKeyFrame, 245, 399
StackPanel, 70, 399
StartIndex, 259
StartupUri, 133
StaticResource, 225
StatusBar, 282
StatusBarItem, 281
Storyboard, 127, 241, 400
Stretch, 52, 121
StretchDirection, 121
Stroke, 317
StrokeThickness, 317
Style, 227, 400
    Complex, 229
    Héritage, 231
    TargetType, 228
Surlignement, 283
Système de coordonnées, 60

T

TabControl, 110, 401
TabIndex, 88
TabItem, 110, 402
Table, 257, 273, 401
TableCell, 273, 401
TableColumn, 273, 401
TableRow, 273, 401
TableRowGroup, 273
Taille, 24
TargetName, 242
TargetType, 228
Template, 232
TextAlignment, 33, 258
TextBlock, 30, 403
TextBox, 38, 404
    Désactivée, 44
    Lecture seule, 43
    Passage à la ligne, 40
    Tabulation, 40
TextDecoration, 35
Texte
    Coupure automatique, 31
    Décoration, 35
    Gras, 34
    Incomplet, 32
    Italique, 34
    Souligné, 34

TextRange, 279
TextTrimming, 32
TextWrapping, 32, 40
TickFrequency, 114
Ticks, 114
TileMode, 51
Title, 135
ToolBar, 183, 405
ToolBarTray, 186, 405
ToolTip, 107
Topmost, 138
Transformation
  Echelle, 221
  Matrice, 222
  Oblique, 221
  Rotation, 220
TranslateTransform, 245, 406
TreeView, 212, 406
TreeViewItem, 406
Trigger, 127, 139, 346, 406
  EventTrigger, 245, 348, 377

U

UIElement, 279
Underline, 34
Uniform, 407
UnloadedBehavior, 126
UpDirection, 324

V

Value, 230, 246
VerticalAlignment, 26, 64, 170
VerticalAnchor, 268
VerticalContentAlignment, 25
VerticalPageSpacing, 254
VerticalScrollBarVisibility, 43, 78
ViewBox, 121, 407
  Stretch, 121
  StretchDirection, 121
Viewport, 51
Viewport3D, 323, 407
Visibility, 176
Visual, 279, 363
Visual Studio, 290

W

WBA, 140, 142
Width, 24, 49
Win32, 13
Window, 135, 408
WindowHeight, 146
Windows Presentation Fundation, 13
WindowsStartupLocation, 135
WindowState, 136
WindowStyle, 137, 342
WindowWidth, 146
WinFX, 13
Wizard, 158
WPF, 13
WPF/E, 165
WrapPanel, 68, 409
Wrapping, 31, 41

X

X
  Key, 227
  TypeArguments, 151
X1, 317
X2, 317
XAMLPad, 17
  Auto Parse, 18
  Cacher le code, 18
  Imprimer, 18
  Police, 18

Refresh, 18
Zoom, 18
XML, 14
Attribut, 14
Balise, 14
Nœud, 14
XmlDataProvider, 207, 409
Source, 209
XPath, 209
XmlStreamStore, 285
XPath, 209

Y

Y1, 317
Y2, 317

Z

ZAM 3D, 313
Zoom, 254

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Composé en France par Jouve
11, bd de Sébastopol - 75001 Paris